APPLICATION EXAMPLE: A 68000 CPU INTERFACING WITH A 673103 DRAM CONTROLLER

As mentioned earlier, no attempt was made to handle system-dependent handshake and arbitration functions. These functions may be easily implemented using programmable logic devices such as PALs(R)" and/or discrete logic devices. The simple logic of the 67310X makes interfacing easy and gives the system designer better control over the output signals' timing. The multiple CASIN-CAS channels are connected to the system's byte data strobes for individual byte access. Doing away with the logic required to split a single CAS output contributes to better skew control as well as to a reduced chip count. A design example shows that less "glue" logic is required to interface the 67310X to a common CPU than is required to interface the more complex "single-chip" controller.

Figure 4 shows a dynamic RAM array controlled by a 673103 interfaced to a MC68000/68010 CPU. The CPU basic system includes an address decoder that selects the different addressing spaces. In this design example, the 673103 operates in its Auto-Access mode which provides on-chip RAS-cas timing. A hidden refresh scheme is implemented in the interface to minimize CPU wait-states due to memory refresh. Two programmable Array Logic PAL devices are used to interface the 673103 to the CPU. One PAL device functions as a system clock divider (RFCXGEN) and provides a refresh clock, while the other PAL device (INTPAL) performs all arbitration and handshake functions.
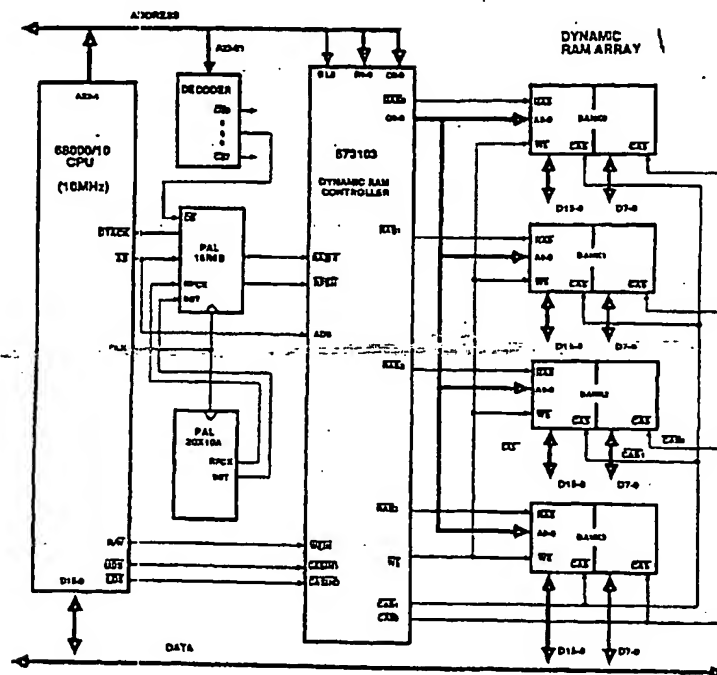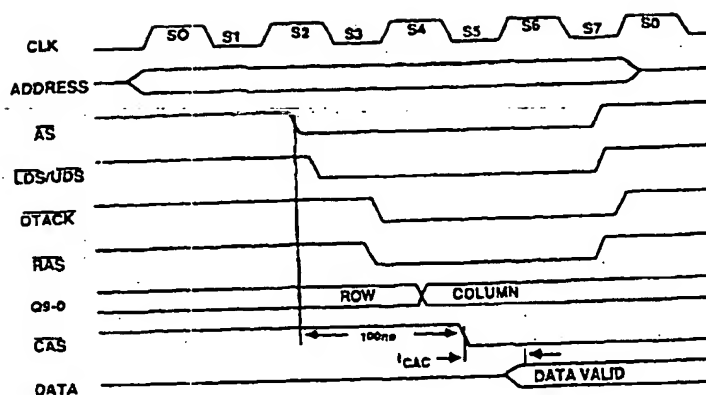

Figure 4. A 673102/3 Dynamic RAM Controller interfacing with a 68000/10 CPU

4/3

CLK, ADDRESS, $\overline{AS}$, $\overline{LDS/UDS}$, $\overline{DTACK}$, $\overline{RAS}$, Q9-0 (ROW / COLUMN), $\overline{CAS}$, DATA (DATA VALID)

FOR THIS SYTEM, THE TOTAL $\overline{AS}$ TO $\overline{CAS}$ DELAY IS 100NSEC. THE DATA IS AVAILABLE 100NSEC + t_{CAC} AFTER $\overline{AS}$ IS ASSERTED, ALLOWING OPERATION AT 10 MHZ WITH NO WAIT-STATES USING120nS DYNAMIC RAMS).

Figure 5. 68000–673102/3 Strobe to Data Delays

The RFCKGEN PAL device is a programmable clock divider which produces Refresh Clock (RFCK) signal. Both cycle and HIGH time of the RFCK are pin selectable, making this particular PAL pattern useful in a range of applications. For this interface, the RFCK determines the rate in which the dynamic RAMs are refreshed (typically 15us). One row of each dynamic RAM is refreshed during the RFCK cycle, thereby refreshing all rows every 2ms . In order to limit power consumption due to the refresh cycles, only one refresh cycle is performed per RFCK cycle. The INTPAL PAL device perform all the required handshake and arbitration functions. It takes in the CPU control signals and the RFCK signal, and initiates DRAM access, DRAM hidden and forced refresh, and interfaces with the system. The hidden refresh scheme takes advantage of CPU memory access cycles which access the static RAMs or EPROMs. When the RFCK is HIGH, and the CPU accesses SRAM or EPROM, a hidden refresh may be performed. This condition is detected when the CPU's Address Strobe (AS) is LOW, but Chip Select (CS) coming out of the address decoder, is HIGH. When the CPU continuously accesses the memory space addressed by the controller, hidden refresh cannot occur. In this case, once RFCK goes LOW, the interface circuit allows the ongoing access cycle to terminate and then initiates a refresh cycle. During refresh, the interface delays any request from the CPU for memory access, by holding the Data Acknowledge

(DTACK) HIGH until the refresh is completed, and RAS precharge requirements are met.

A memory access cycle starts when AS is asserted (pulled LOW) by the CPU, and terminates once the interface circuitry responds by asserting the Data Acknowledge (DTACK) signal. During a read cycle, data must be available to the CPU no later than 185ns (for 10MHz 68000/68010 CPU) after AS goes LOW 0 no-wait-states operation is attempted. In this design example, the data is available 100ns + t_{CAC} after AS goes LOW (t_{CAC} , CAS access time, is a DRAM parameter). Therefore no-wait-states operation can be achieved using 120ns dynamic RAMS with t_{CAC} of 60ns or shorter, leaving enough margin for buffer delays (see Figure 5). An early write cycle is used, and data is available to the memory at least 20ns before CAS goes LOW. Data is maintained at least 20ns after CAS goes HIGH. The described design is 40ns faster than a similar design using a single-CAS controller to control the dynamic RAM. Furthermore, this design requires at least one less chip than designs using other controllers. This design was built and tested at 10MHz operating with no wait-states and using 120ns dynamic RAMs. Detailed schematic and PAL device specifications can be obtained from the authors.

5

Data Bits	Check-Bits	Overhead Percentage
16	6	37.5%
32	7	21.9%
64	8	12.5%

Table 2. Modified Hamming Code check-bit overhead

ERRORS IN DYNAMICS STORAGE

The physical dimensions, the signal level and the stored charge of the dynamic memory cells are greatly reduced to allow denser DRAM IC's. As the stored charge in the DRAM cells decreases, the device is more susceptible to soft errors caused by alpha particles as well as by environmental noise.

Soft errors are temporary, random, and they can be corrected by rewriting into the erroneous cell. Therefore, if the system has the ability to locate the bit-in-error, the soft error can be corrected. One method to locate the bit-in-error is to append a fixed number of check-bits to the data word and to store these check-bits along with data during memory write. Upon reading, both the data and check-bits are read into the EDAC. The check-bits are regenerated from the read data, and these newly generated check-bits are compared with the read check-bits. The comparison produces the syndrome bits. The syndrome bits constitute a binary number that points to the erring bit. This encoding scheme of generating the check-bits and the syndrome bits is called Hamming code, proposed back in 1950 by Hamming.

The modified Hamming code encoding scheme has one more check-bit overhead than the original Hamming code does, but it can detect all double-bit errors as well as detect and correct all single-bit errors. In addition it can detect a substantial number of multiple-bit errors.

With the introduction of the 1 megabit DRAM from various semiconductor memory manufacturers, the need to protect the integrity of the memory system has never been greater. The modified Hamming code has proved particularly useful in the application for its relatively low overhead in today's wide data words system and its capability to detect and correct single-bit errors and detect all double-bit errors. (See table 2)

EDAC ARCHITECTURES IN SYSTEMS

The system performance depends greatly on the EDAC architecture. In general, there are two kinds of architecture that exist in EDAC IC's.

into the system because of its I/O requirements. The second architecture is the Flow-Through architecture. This architecture is recently reintroduced in the MMI's 67C3290 (32-bit) and the MT's 3C0038 (36-bit). These Flow-Through EDAC devices feature separate data in and data out ports for the read cycle. This architecture simplifies the memory system design thereby improving overall system performance.

BUS-WATCH

For the Correct-Always configuration in the Bus-Watch architecture, data is checked and corrected if necessary before it is placed on the bus. In this configuration, the EDAC is placed parallel with the data bus. Data from CPU goes to both the memory and the EDAC; the EDAC generates check-bits and stores them along with the data word during a memory write cycle.
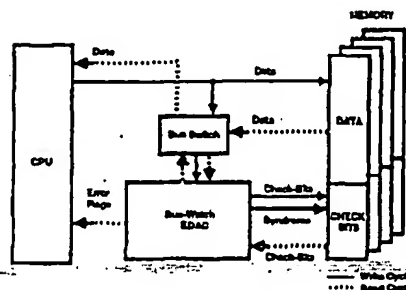
FIG 6. Correct-Always mode in Bus-Watch architecture

In a memory read cycle, data from memory is read into the EDAC to generate new check-bits; at the same time, check-bits stored in memory are read into the EDAC. The read check-bits are compared against the newly generated check-bits, thereby producing syndrome bits. The syndrome bits indicate whether the data word has no error, single-bit error, double-bit error. In the case of no error, data is placed on the data bus for system usage; in the case of single-bit error, the EDAC inverts the bit-in-error and places the corrected data on the data bus; for double-bit error, no correction is attempted, only the multiple-error flag is asserted.

To eliminate the bus-switching circuitry in the Bus-Watch architecture, the EDAC configuration in the system can be changed slightly: the EDAC is attached to the data bus so that the system can run as fast with the EDAC as without. However, the EDAC can only flag the host for errors, it cannot correct the erroneous data. This configuration is called Detect-Only. For most systems, this Detect-Only configuration is unacceptable because the erroneous data has already been processed when the error flag interrupts the CPU. (See figure 7)

FLOW-THROUGH

The second EDAC architecture is the Flow-Through architecture. The MMI's 3290, a 32-bit EDAC, is based on this architecture. The innovative design and novel architecture on the EDAC IC improves the system's performance, at the same time it eliminates the bus-switching circuitry, simplifies the design, reduces chip-count and saves board space.
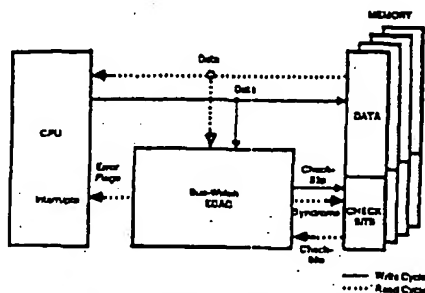

FIG 7. Detect-Only mode in Bus-Watch architecture

In the Flow-Through architecture, the EDAC device is placed in the data path between the CPU and the main memory. In a memory write cycle, data is written to the data section of the memory; it is also written to the EDAC through a bidirectional port acting as inputs. The EDAC generates the check-bits according to the modified Hamming code and presents these check-bits at the check-bit outputs, to be stored along with the data into the check-bit section of the memory. (See figure 8)

In a memory read cycle, data from the data section of the memory is read to the EDAC through a data input port. Concurrently, the stored check-bits from the check-bit section of the memory are read to the EDAC through the check-bit inputs. The EDAC uses the read data to generate new check-bits and compares the newly generated
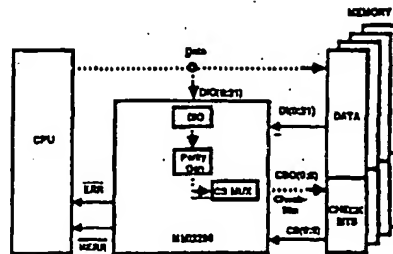

FIG 8. Write Cycle in Flow-Through Architecture

check-bits with the read check-bits. The syndrome bits points to the erring bits in the data word. The error flags respond accordingly.

When the Correct-Always mode is selected, and if there is a single-bit error, the syndrome bits point to the erring bit and the data is passed through the correction logic to correct the data; the corrected data is then placed on the data bus through the bidirectional port acting as outputs.

To provide maximum flexibility for the users, the Detect-Only can be configured easily by deselecting correction. The data flow follows the Correct-Always mode, except that the read data is placed on the data bus, uncorrected, bypassing the correction logic.

Notice that there is no hardware modification involved in switching between modes. (See figure 9)
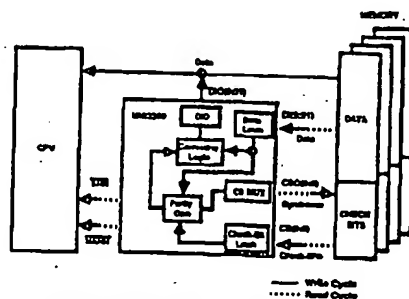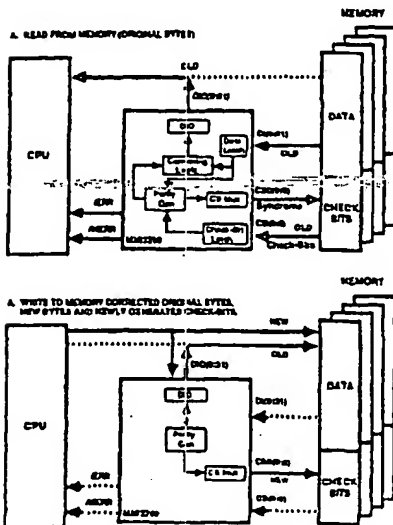

FIG 9. Read Cycle in Flow-Through Architecture

7

4/3

BYTE-WRITE AND SCRUB CYCLES

Byte operations are increasingly popular in microprocessors, especially in today's microprocessors with wide-data words. The MMI's 3290 is designed with the intent of keeping byte operations as simple as possible. The byte write cycle starts with a read from memory. Data and check-bits are latched in the EDAC. When the control changes from read to write, the syndrome bits are latched in the EDAC to maintain the unchanged data bytes, at the same time the new data bytes from the CPU are written into memory along with the unchanged data bytes. Check-bits for the modified data word are generated and stored along with the modified data word. (See figure 10)

Another useful cycle that the EDAC offers is the scrub cycle. In normal operation the EDAC corrects single-bit error on the fly but the error is not corrected in the memory. A double-bit error occurs when a single-bit soft error is left uncorrected and then comes along another single-bit soft error at the same memory location. Scrubbing memory will avoid such double-bit errors which are uncorrectable by the EDAC. The scrub cycle is initiated by a correct read. The corrected data is presented at the data bus. The corrected data, the check-bits and the syndrome are latched in the EDAC to maintain the corrected data on the data bus. The control then switches to a write to store the corrected data word and the new check-bits into the memory. (See figure 11)

A. READ FROM MEMORY (ORIGINAL BYTE II)

A. WRITE TO MEMORY CORRECTED ORIGINAL BYTES, NEW BYTES AND NEWLY GENERATED CHECK-BITS.

A. READ FROM MEMORY

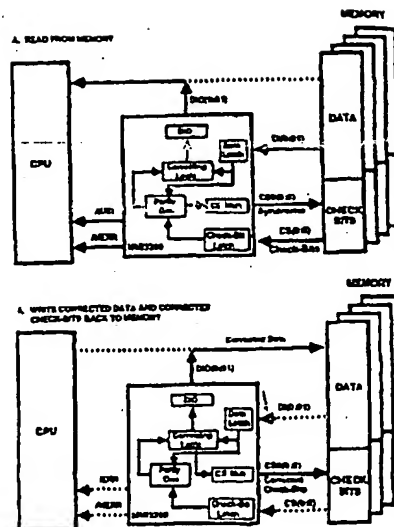A. WRITE CORRECTED DATA AND CORRECTED CHECK-BITS BACK TO MEMORY

FIG 11. Scrub Cycle

In addition to the flexible cycles provided, the EDAC also has elegant diagnostic cycles. The diagnostic cycles allow the check-bits to be latched in the EDAC under external control. These check-bits can be written into memory in a diagnostic write cycle along with erroneous data . These latched check-bits can also be used to generate the syndrome bits in a diagnostic read. These two cycles allow the designer the flexibility of implementing a variety of diagnostic sequences. The designer can force a known pattern of check-bits into memory along with non-corresponding data by latching the check-bits in the EDAC and writing these check-bits together with an erroneous data word into memory. That way an error is inserted. The memory may be read back via a correct read to check the operation of the EDAC.

The designer can also latch the check-bits in the EDAC and starts a diagnostic read. The EDAC uses the latched check-bits and the read data word to generate the syndrome bits. The latched check-bits with the syndrome, or the latched check-bits with the newly generated check-bits are placed on the data bus for verification.

MMI has yet another EDAC based on the modified Hamming code. The MMI's 3291 is the expandable version of the MMI's 3290. Two MMI's 3291 connect together to support 64-bit data word in which one part is the master and the other is the slave.

The slave part calculates the partial parities for the least significant 32 data bits. These partial parities go to the master part and combine with the partial parities from the most significant 32 data bits. The master part then generates the check-bits for the full 64-bit data word. Both versions are fabricated using the 1.75 micron CMOS technology.

CONCLUSION

The new DRAM controllers' architecture, as well as tight skew specifications and short propagation delays contribute to fast DRAM access times and thereby to overall system performance. The novel architecture cuts chip count and simplifies the logic to streamline the system design.

The Flow-Through architecture clearly is superior to the Bus-Watch architecture in terms of design simplicity and system performance. With separate ports for data input and output, the Flow-Through EDAC simplifies the task of the designer considerably. The bus-switching hardware as well as its control logic are eliminated. The state machine, used to provide the proper sequence of control signals for all the components of the memory board, is also simplified a great deal.

From the system point of view, using the Flow-Through EDAC improves the overall system performance significantly. In the Flow-Through EDAC, the propagation delay associated with the bus-switching circuitry is removed. Also, the time for data to flow through the Flow-Through EDAC is less than the one of the Bus-Watch EDAC because the enable, disable and recovery time of the single port I/O is eliminated. In addition, chip-count is reduced in Flow-Through EDAC system, thus the designer saves board space and reduces cost.

9

4/3

DESIGN ENTRY

100-MHz DRAM controller
sparks multiprocessor designs

Naseer Siddique
Signetics Corp., 811 E. Arques Ave., Sunnyvale, CA 94088-3409; (408) 991-2000.

As computer designs with a single CPU give way to those with multiple processors, designers must tackle the problem of processors jockeying for system memory. That solution proves even more elusive when the memory is made of dynamic RAM, as is most often the case. Of the choice between static and dynamic RAMs, static RAMs are fast but they are also expensive while dynamic RAMs better suit multiprocessor systems, which require a large number of memory chips.

A fast dynamic RAM dual-ported controller saves space, improves reliability and offers a choice of latched and unlatched address lines.

Though the need for dynamic RAMs in most large multiprocessing systems is obvious, most dynamic RAM controllers are one-port devices. The few that have two ports are slow. Recent exceptions to that state of affairs are the 74F764 and 74F765 dual-port controllers, which guarantee a 100-MHz clock frequency. This speed permits control of 40-ns dynamic RAMs.

Besides saving board space, the new devices improve system reliability and cut down on design and debugging time. The 764 differs from the 765 in that it has an on-board address input latch. The latch is useful in systems that do not have their own.

The controllers have logic to request refresh cycles, arbitrate memory access, multiplex addresses, and generate timing signals—functions that would otherwise require about 25 discrete devices. Each directly drives well over 100 dynamic RAMs without external buffers. A proprietary circuit ensures that switching occurs on incident waves rather than on reflected waves—an important requirement in high-speed operation.

Each controller (Fig. 1) is a synchronous device, with all signal timing and control signals generated in step with the input clock, CP. A refresh clock input, RCP, sets the refresh period for each row. For memories that have their own refresh circuits or that do not require any, RCP is left in the high state. Each refresh request increments a counter, which addresses the memory during the refresh cycle.

The arbitration logic has two stages. The first stage decides which of two request inputs, REQ_1 or REQ_2, the chip will respond to. Depending on the choice the logic asserts a corresponding select output, $\overline{SEL_1}$ or $\overline{SEL_2}$. Since the $\overline{SEL}$ outputs select one of two external devices that access the memory, this output can indicate which processor's address bus should be asserted at the controller's address inputs. Arbitration takes place whether or not a refresh cycle is already under way.

The second stage of arbitration selects between the selected processor and internal refresh requests. Since refresh requests take priority, they are serviced immediately after a current cycle ends.

The arbitration logic also generates a grant output signal, GNT, at the start of every memory-access cycle. The GNT, $\overline{SEL}$, and data transfer acknowledge (DTACK) outputs generate wait states, if needed, by a fast processor.

If the two processors simultaneously request access to a dynamic RAM, the controller resolves the contention with arbitration logic that samples request inputs REQ_1 and REQ_2 on different edges of the CP clock: the logic samples REQ_1 on the rising edge of the CP clock and REQ_2 on the falling edge (Fig. 2). Special flip-flops in the logic greatly reduce the likelihood of metastable states.

When a processor requests access to the memory (by asserting the $\overline{REQ}$ input) and neither a refresh cycle nor the other request input is active, the controller asserts the $\overline{SEL}$ output that corresponds to the active input $\overline{REQ}$. A GNT output then goes high to mark the start of a memory access cycle. On the other hand, if a refresh cycle is already in progress, the $\overline{SEL}$ output is asserted but the GNT output stays inactive until the cycle completes.

A third possibility is that the controller is already handling a memory access cycle. In that case, $\overline{SEL}$

for the other processor is not issued (though GNT is already active), ensuring that no contention occurs on the address bus, that is, the address bus is not driven by two processors at the same time. When the memory access cycle ends, the controller asserts the $\overline{SEL}$ output corresponding to the waiting $\overline{REQ}$ input. If no other refresh request is pending, a GNT output follows; if a refresh is waiting, the controller responds to it.

When the GNT output goes high, the 74F764 latches address input A_1 to A_{18} and send signals A_1 to A_9 to its memory address pins MA_0 to MA_8. The 74F765, of course, does not latch the inputs, but sends them directly to memory-address outputs MA_0 to MA_8. The chip awaits one-half of a clock cycle for the address signals to propagate through to the outputs, after which it asserts its row address strobe ($\overline{RAS}$) output.

One clock cycle later, the 74F764 selects the latched address lines A_{10} to A_{18} and sends them to its memory-address outputs. Since the 74F765 did not latch the address inputs, it passes address lines A_{10} to A_{18} directly through to the memory-address outputs. At the same time, the controller asserts a write gate output, WG. That output gates the write-strobe pulse from the selected processor to start an early write cycle. Just as for row addresses, the controller allows one-half of a clock cycle for A_{10} to A_{18} to propagate and stabilize. Then it asserts its column address strobe enable, CASEN, which can serve as a $\overline{CAS}$ output or be decoded with the higher order address bits to produce multiple $\overline{CAS}$ signals.

The WG output may serve as a select signal for multiplexing additional address lines for 1-Mbit and larger dynamic RAMs. In this case, additional external refresh address bits may not be needed because the controller has a 9-bit refresh address counter (512 row addresses). This exceeds the refresh needs for industry-standard 1-Mbit dynamic RAMs.

After the controller asserts CASEN, it waits for two-and-a-half clock cycles and then negates $\overline{RAS}$, making the total $\overline{RAS}$ pulse width four clock cycles. Since this width matches the standard dynamic RAM access time, the controller next asserts DTACK output, indicating that valid data is on the dynamic RAM data lines or that an access cycle is complete. DTACK is useful for driving processors that require such acknowledgement. Alternatively, the GNT output generated at the start of a memory-access cycle can also acknowledge completion and without incurring wait states.

All output signals stay in their final state until the selected processor withdraws its request, which it does by negating the corresponding $\overline{REQ}$ input. After the request is withdrawn, the controller synchronizes its internal signals, negates its output pulses, and attends to any pending requests or refresh cycles. The controller starts a refresh cycle by sending the output signals of a nine-bit refresh counter to the MA_0-MA_9 outputs. After one-half of a

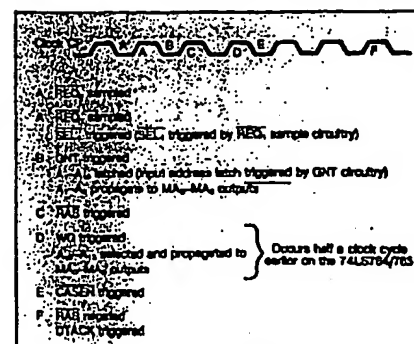clock cycle, the controller asserts the $\overline{RAS}$ output for four clock cycles. Then it negates them for three clock cycles to meet the dynamic RAM's $\overline{RAS}$ precharge requirements.

The dual-port controllers flex into a wide range of configurations. For example, they can help 8085 microprocessors share a 16k-by-8-bit (eight 16k-by-1-bit devices) dynamic RAM (Fig. 3). The data and the eight least significant address bits on the 8085 are multiplexed. As a result, these lines make good use of the controller's input latch.

When either of the two processors asserts address latch enable (ALE), external decoding circuitry decodes the respective address lines and requests a memory access by asserting an $\overline{REQ}$ input. In response to $\overline{REQ}$ the controller
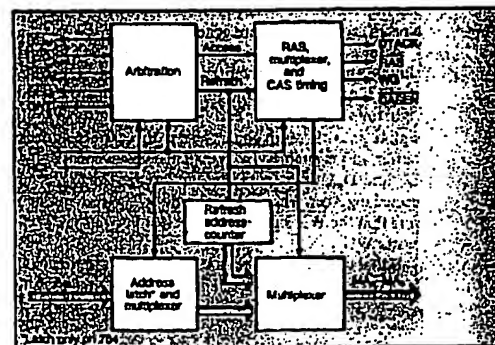
1. The 74F764/765 is the fastest dynamic RAM dual-ported controller, with a guaranteed clock frequency of 100 MHz. On-board arbitration and timing logic does the work of 25 discrete devices.

A. $\overline{REQ}$ sampled
$\overline{REQ}$ sampled
$\overline{SEL}$ asserted ($\overline{SEL}$ triggered by $\overline{REQ}$ sample circuitry)
B. GNT triggered
A_1-A_{18} latched (input address latch triggered by GNT circuitry)
A_1-A_9 propagate to MA_0-MA_8 outputs
C. $\overline{RAS}$ triggered
D. WG triggered
A_{10}-A_{18} selected and propagated to MA_0-MA_8 outputs
E. CASEN triggered
F. $\overline{RAS}$ negated
DTACK triggered

Occurs half a clock cycle earlier on the 74LS764/765

2. By sampling request signals $\overline{REQ}_1$ on the rising edge of the CP clock and $\overline{REQ}_2$ on the falling edge, the controller resolves any contention when both processors seek access simultaneously.

generates the corresponding $\overline{SEL}$ output, enabling the associated three-state address and data buffers. Because the read ($\overline{RD}$) and write ($\overline{WR}$) signals coming out of the 8085 cannot be true at the same time, only the $\overline{WR}$ strobe controls the direction control (S/$\overline{R}$) signal on the data buffers.

External gating sends wait states to the processor enabling the controller to arbitrate among the two $\overline{REQ}$ inputs, any refresh requests, and the completion of the current cycle. The controller's GNT output terminates the wait state. If neither a refresh nor an access cycle is occuring, the controller asserts $\overline{SEL}$ and GNT outputs within two cycles of the CP clock. If the controller is clocked at 30 MHz and the 8085 runs at 5 MHz, $\overline{SEL}$ and GNT are asserted before the Ready line is sampled by the CPU. As a result, no wait states are generated.

After a memory-access cycle begins, the 764's internal timing and control circuitry automatically generates $\overline{RAS}$, $\overline{CASEN}$, WG, and multiplexes addresses at the right time. To achieve an early write cycle, the CPU's $\overline{WR}$ signal goes through the external three-state address buffer and is enabled by WG. This arrangement allows D_{in} and D_{out} lines from the dynamic RAM to be connected together. The controller's clock, however, must ensure that the $\overline{WR}$ signal is valid before the controller generates WG.

If that condition on the clock slows access time enough to cause wait states, a designer can invert $\overline{RD}$ and use it as

the $\overline{WR}$ signal. Another choice is to separate the D_{in} and D_{out} lines. In that case, the D_{out} lines are enabled through three-state buffers by $\overline{RD}$. The delayed $\overline{WR}$ signal produces a late write cycle and is independent of the controller's WG signal.

A second application connects two 68000 microprocessors to a 1-Mbyte dynamic RAM consisting of two banks, each of sixteen 256k-by-1-bit devices (Fig. 4). Since the microprocessors' address and data buses are not
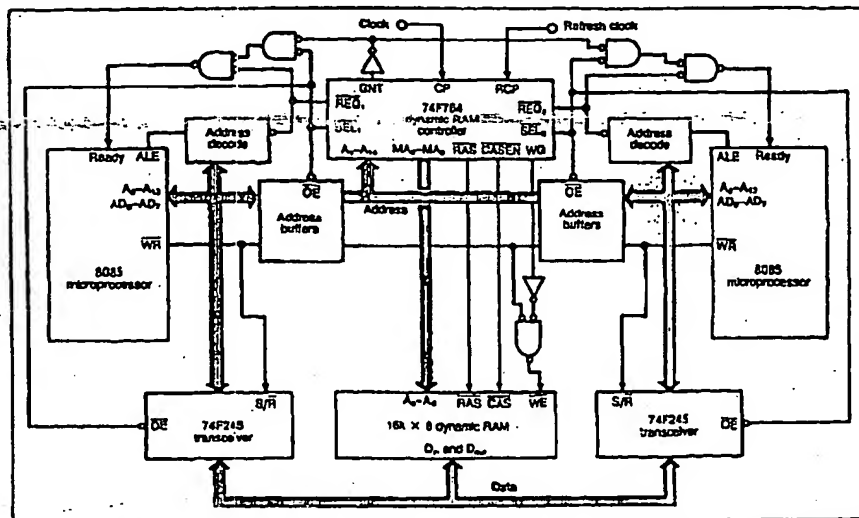
3. The 8085 multiplexes data and the least significant address byte. As a result, this task would usually call for external latching. The controller's address-input latch, however, makes external latches unnecessary.

multiplexed, the nonlatching 74F765 is adequate.

Memory bank A consists of upper data byte A, UDBA, and lower data byte A, LDBA; bank B is upper data byte B, UDBB, and lower data byte B, LDBB. A 74F139 decoder decides which bank to access by decoding address bit 19, A_{19}, of the 68000 and $\overline{CASEN}$ from the controller. The 74F139 generates multiple CAS signals to the dynamic RAM chips. At any given time, CAS is asserted to either bank A or bank B.

The data byte access depends on the microprocessors' upper data strobe ($\overline{UDS}$) and lower data strobe ($\overline{LDS}$), which further decode the 74F139 outputs. The overall effect is to assert $\overline{CAS}$ to one or both of the 256k-by-8-bit RAMs within a bank. For 16-bit transfers, $\overline{UDS}$ and $\overline{LDS}$ are asserted at the same time, allowing simultaneous access to UDBA and LDBA.

Respective $\overline{SEL}$ outputs from the controller and $\overline{UDS}$ and $\overline{LDS}$ enable one or both of the data buffers that correspond to a selected processor. That processor's $R/\overline{W}$ strobe controls the direction of data flow through the buffers. Additional gating circuits ensure that DTACK is active only for the selected processor and only after being asserted. Decoding the address bus when AS is asserted generates the $\overline{REQ}$ inputs from both processors. □

Naseer Siddique has been an application engineer for LSI products in Signetic's Standard Products Division since 1983. He holds a BSEE from the University of Engineering and Technology in Lahore, Pakistan, and an MS in Computer Engineering from Wayne State University in Detroit.

How valuable?	Circle
Highly	559
Moderately	560
Slightly	561
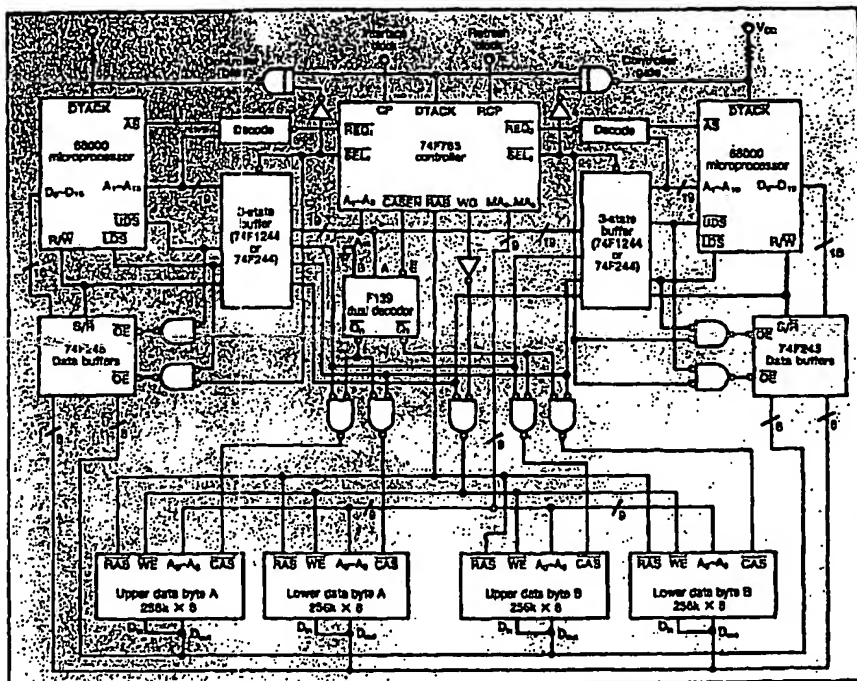
4. The memory controller lets two 68000 microprocessors share one main memory. Decoding of $\overline{CASEN}$ from the controller and address bit 19, A_{19}, of one of the 68000 microprocessors determines which of the two memory banks is accessed. Then, the 68000's upper and lower data strobes determine which data byte is accessed within the bank.

A 4-MBIT CMOS SRAM WITH 8-NS SERIAL-ACCESS TIME

Hirotada KURIYAMA, Toshihiko HIROSE, Shuji MURAKAMI, Tomohisa WADA,
*Koreaki FUJITA, Yasumasa NISHIMURA, and Kenji ANAMI

LSI R. & D Laboratory, *Kita-Itami Works, Mitsubishi Electric Corporation
4-1 Mizuhara, Itami, Hyogo, 664 Japan

1. Introduction

Recent image-processing systems demand high speed serial access memory. As for a 16-Mbit DRAM, the 10-ns serial access architecture has already been reported[1]. However, the architecture can't access up to 2-kbits serial of 16-Mbits.

This paper describes an 8-ns serial access architecture newly embedded in a 4-Mbit SRAM, which can access up to 4-Mbits. This memory realizes a 125-MHz fast serial READ/WRITE operation suitable for ultra high speed memory systems such as an image processing system, a high speed testing system and super computers. This function is also beneficial for reducing testing time of the RAM.

2. Serial Mode Circuit

Figure 1 shows a block diagram of the RAM focusing on serial mode circuits. A 4-Mbit memory cell array is divided into 32 blocks. In addition to the conventional architecture, four kinds of hierarchical shift registers (Data Bus SR, Transfer Gate SR, Sense Amplifier / Write Driver SR, Row SR), Row Address Counters, Data Bus Selector, Look-Ahead Row Decoder and Serial/Normal Controller are added for continuous serial READ/WRITE operation. Both external signals /SE (serial enable) and CLK (CLK is also used as an address in normal operation) control above serial circuits.

Timing diagram of serial READ cycle is shown in Figure 2. After /SE signal goes low, 40-ns initializing action (setting the first address in the registers) starts. Then the serial operation is performed by the external clock (CLK).

Figure 3 shows the block diagram of the serial mode in regard to the Sense Amplifier(SA). A block of the memory cell array is composed of 128 columns, which is further divided into 16 sub-arrays, and each sub-block array has a pair of Transfer Gate (TG) and SA. These TG and SA are broken down into two groups (group-A and group-B), and are controlled by the Transfer Gate Shift Register (TGSR) and Sense Amplifier Shift Register (SASR), respectively. The output data of 16 SA's are transferred to the Data Output through the Data Bus Selector which is controlled by the Data Bus Shift Register (DBSR). The column addresses are increased by shift registers DB,TG,and SA in this order. The hierarchical registers reduce the number of registers compared with the conventional register array. The serial access time is determined by only DB selection, which is the lowest address selection in the serial access mode.

In order to achieve the same fast serial access time at the change of TG selection, the SA output of both group-A and group-B are transferred alternatively to Data Output every two cycles (e.g. group-A during cycle #0, 1, 4, 5, etc., group-B during cycle #2, 3, 6, 7, etc.). Timing diagram of the TGSR is shown in Figure 4. The outputs of TGSR in group-A vary two cycles earlier than that in group-B. These two cycles (a) are used for the change of TG in group-A, then the change of TG in group-B is performed while group-A is selected. Consequently, the same fast serial access time with no break has been realized at the change of TG with thus interleaved

circuit. Figure 5 shows a timing diagram of the SASR and Row SR (RSR). The SASR changes the block with the same interleaved method (b) as above. The RSR is a new shift register with overlapping selection period(c).

The word line select circuits at the block #0, 1 are shown in Figure 6. In order to start the serial operation at any address, special word line select circuits (two word line switches, Look-Ahead Row Decoder and Row Address Counters) are newly applied. Only memory cell block #0 has two word line switches and two kinds of row decoder (Look-Ahead Row Dec. on the left and Normal Row Dec. on the right) for changing the word line operation according to the modes. When the uppermost block #31 is selected in the serial mode, the Look-Ahead Row Dec. prepares the selection of the next row address with overlapping period. Therefore, the same fast serial access time has been realized up to 4M-bits.

3. Characteristics

Figure 7 shows a chip photomicrograph of the RAM. The chip size is 8.0 mm x 18.35 mm. The area penalty of serial circuit is about 8 %. The serial access time of 8-ns is obtained at typical condition with 3.3V supply voltage. The typical characteristics of the RAM are summarized in Table I.

4. Conclusion

An 8-ns serial access time has been realized in a 4-Mbit Static RAM with the newly proposed circuits (hierarchical shift registers and Look-Ahead circuits),which can access up to 4M-bits. This serial function achieves a 125-MHz fast serial READ/WRITE operation suitable for ultra high speed memory systems. This function is also beneficial for reducing testing time of the RAM.

Reference

[1] S.Watanabe,et al,"An Experimental 16Mb CMOS DRAM Chip with a 100MHz Serial Read/Write Mode".ISSCC DIGEST OF TECHNICAL PAPERS, p. 248-249; Feb.,1988.
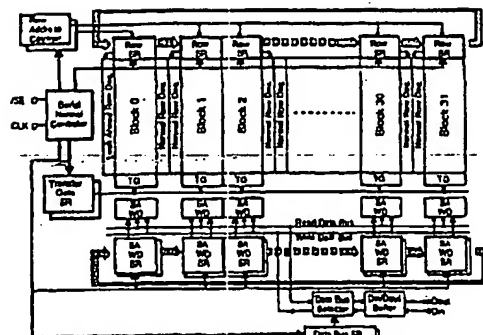
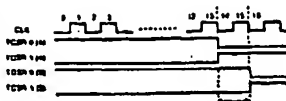Fig.1 Block diagram of RAM focusing on serial mode circuits
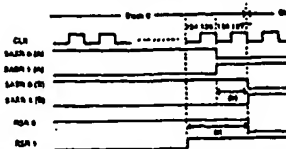

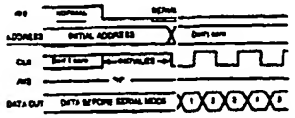Fig.4 Timing diagram of TGSR

Fig.5 Timing diagram of SASR and RSR
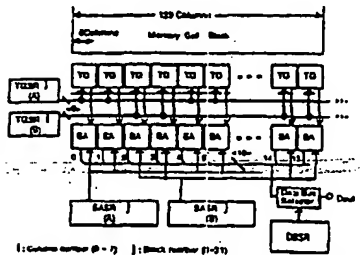

Fig.2 Timing diagram of the serial READ cycle

Fig.6 The word line select circuits

I: Column number (0 – 7)   J: Block number (1–31)

Fig.3 Block diagram in regard to Sense Amplifier
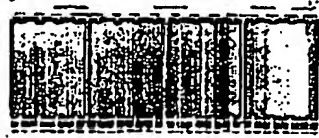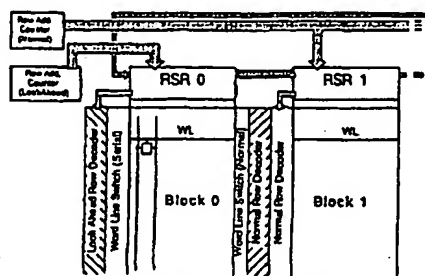
Table I RAM characteristics

Organization	4Mx1 / 1Mx4
Serial Start Address	Arbitrary Address
Serial Read/Write	Up to 4Mbit
Control Pin	/SE,CLK (Address Pin)
Initial Time	40 ns
Access Time	8ns (Serial Mode)
	20ns (Normal)
Process	4poly-Si,2Al
	0.8μm (NMOS) ,0.8μm (PMOS)
Cell Size	3.5μm x 5.3μm
Chip Size	8.35mm x 18.0mm

Fig.7 Chip photomicrograph

1990 SYMPOSIUM ON VLSI CIRCUITS

DIGEST OF TECHNICAL PAPERS

1990 **VLSI**

**CIRCUITS
SYMPOSIUM**

HONOLULU

A 1.2 ns GaAs 4K Read Only Memory

Jinc Chun, Richard Eden, Alan Fiedler, Daniel Kang, and Lap Yeung

GigaBit Logic, 1908 Oak Terrace Lane, Newbury Park, California 91320

ABSTRACT

The first commercially available GaAs 4K ROM has been designed and manufactured using GigaBit Logic's 3-level metal HMED (High Margin Enhancement/Depletion) Process. The access time of 1.2 ns is obtained with a power dissipation of 1.9 W. The part is ECL compatible, and packaged in GigaBit Logic's standard 40 pin package.

INTRODUCTION

In today's high speed environment, an IC component that can provide ultra fast look-up table capability is a must for many digital applications. GaAs RAMs are available for such applications, but RAMs are slower and would require additional hardware to load necessary functions into the RAMs. Application of the digital look-up table ranging from the direct digital synthesis to custom logic functions can benefit from a mask programmable GaAs ROM that can provide fast turn-around time with several choices of power/speed combination. Table 1 summarizes the features of the 4K ROM.

This 512 X 8 ROM uses a single FET as a ROM cell, and the FET is programmable to be active by using a 2nd via mask. This same mask also programs on-chip output enable decoders which allow memory expansion up to 32 K without the need for external decoding that decreases the system cycle time. Optimum performance in power and speed is achieved by GigaBit Logic's production process, HMED (High Margin Enhancement/Depletion) [1]. The process includes a recessed 1 um gate and makes available two depletion pinch-off voltages to maintain good design margins with high performance. This design uses the three levels of interconncet metal available in this process.
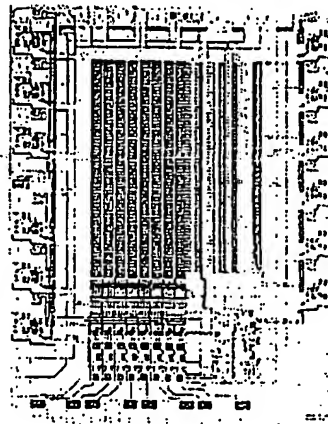

Figure 1
4K ROM Die Photo

Organization	512 X 8
Address access time	1.5 ns max
Output Enable access time	1.0 ns max
Power dissipation	2.0 W max
I/O interface	ECL compatible
Power supply voltage	0 V, -2 V, -5.2 V
Chip size	2.44 mm X 3.55 mm
Package	GigaBit Logic 40 pin LCC
Process	GigaBit Logic HMED
Gate length	1.0 um
Threshhold voltage	-0.6 V, -0.25 V
Interconnect	Triple level metal process

Table 1
4K ROM Characteristics and Fabrication

GaAs IC Symposium - 83

CIRCUIT DESIGN

The basic logic gate used in the peripheral circuit is improved Capacitor Diode FET Logic (CDFL) shown in Figure 2. The performance of such an inverter with resistive power gain load and low power super-buffer have been previously published [2] [3] and proven over the years in GigaBit Logic's designs for their reliable performance.
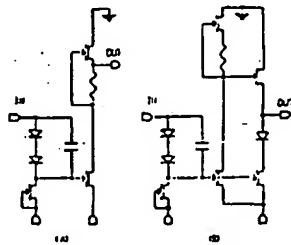
Figure 3 shows the input buffer schematic of the 4K ROM. To achieve high speed performance, one gate delay is eliminated from the conventional input buffer. The differential amplifier is designed to drive the push-pull output stage directly, and the input buffer delay is only 250 ps with 1.0 pf of loading at the output. Using the high margin enhancement FET(Vp = -0.25 V) as a pull-up device in the push-pull keeps the power low for total 12 input buffers on the chip.

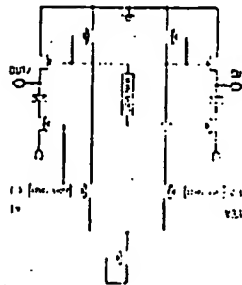The row decoder, shown in the Figure 4, is chosen among the variety of configurations using BFL (Buffered FET Logic), CDFL, and DTL (Diode transistor Logic). The simulation result shows clearly that DTL gives the best performance in speed and power while it requires the least layout area. The DC current through the DTL logic is carefully optimized, so that the critical IR drop in the address bus is within the allowed range over the temperature and process variation. The layout of the row decoder is also critical due to the parasitics and backgating introduced from the small layout space dictated by the single FET ROM cell. The row decoder pitch is 26.4 um. The power consumption of the 63 de-selected row decoders is kept low by using an internally generated power supply that turns off the pull-up FET in the word line driver.
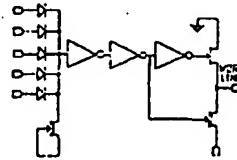

Figure 2
(A) Power gain inverter (B) Low power super-buffer

Figure 4
4K Rom Row decoder

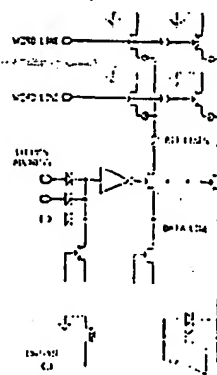Figure 3
4K ROM Input buffer

Figure 5
4K ROM Array Architecture

The column decoding is done by a pass gate scheme that connects one out of eight bit lines to the common data line for sensing (Figure 5). Therefore, the loading for the column decoder is small enough to have only one inverting stage after the DTL decoding. This is essential to eliminate the access time difference between row and column. It takes more time to sense after the column decoding because the selected bit line which is normally precharged to VDDL needs to be discharged to its proper bias voltage for sensing. The layout space is even more limited by the column pitch of the ROM cell, which is 12 um.

The sense amplifier senses the current on the highly capacitive data line rather than the voltage. The feedback diodes limit the swing of the data line within 300 mv, while the output of the sense amplifier is large enough to drive the output source follower FET. The output of the ROM drives a transmission line terminated by a 25 to 50 ohm resistor.

Figure 6 shows the full circuit simulation result of speed and power as a function of Vp. Each circuit in the ROM is optimized with worst case power supplies and Vp variations for the maximum performance in the wide range of power and speed combination. The speed shown in Figure 6 is simulated at 0° C junction temperature and the power is simulated at 125° C junction temprature with most negative operating power supplies rated for the ROM.
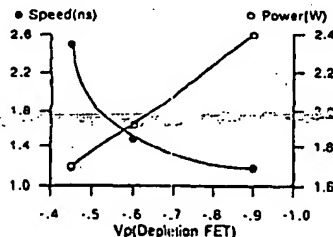
driven by the address generator (12G014) to unload its contents through MUX into GigaBit Logic's 1K RAM (12G014) which operates at 2.5 ns cycle time. The 1K RAM contents are later read into a PC at a slower speed for comparison. A clock frequency of up to 800 MHz has been achieved with this 4K ROM.
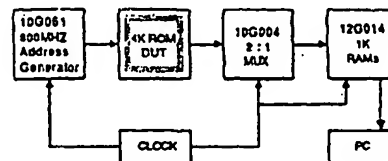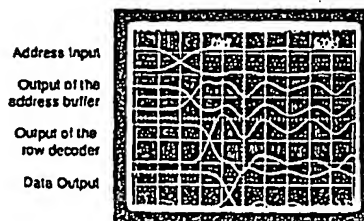

Figure 7
4K ROM High speed tester

Address input

Output of the
address buffer

Output of the
row decoder

Data Output

Figure 8
Internal waveforms (true vertical scale is 1 V/division)

Figure 6
4K ROM Vp vs. Speed and Power

TEST RESULT

The block diagram of the production tester is shown in Figure 7. The custom-built, in-house tester using exclusively GigaBit Logic's standard parts can test 4K ROMs up to 800 MHz. The 4K ROM under test is

Circuit	Delay
Input Buffer	250 ps
Row Decoder	450 ps
Column Decoder	300 ps
Sense Amp / Output Buffer (from the word line change)	500 ps
Sense Amp / Output Buffer (from the column select change)	650 ps
Total address access time	1.2 ns

o Measured at 25° C, nominal power supply
o Input signal; 1 V peak to peak

Table 2
Measured AC performance of 4K ROM

Figure 8 shows the internal waveforms taken at wafer probe. The address access time is 1.2 ns with operating power of 1.9 W at 25° C chuck temperature. The access time further improves to 1.1 ns at 125° C with 2.2 W of operating power. The summary of measured AC performance is listed in Table 2.

Based on the described 4K ROM, the first commercially available direct digital synthesizer has been introduced in the market place in July of this year [4]. In DDS, the 4K ROM is used to store words which give the amplitude of a sine (or cosine) function. A fast accumulator (GigaBit Logic's 10G102) [5] is used to generate ROM addresses, with each address corresponding to a specific phase of the stored sinewave. The ROM's output is digital-to-analog converted and low pass filtered in order to produce a synthesized sinewave with good spectral purity. Typically, the maximum synthesized frequency can be as high as 250 to 400 MHz (at Nyquist rate which is 45 % of the actual clock frequency). The 4K ROM also constitutes the fastest 9-input, 8-output combinatorial PLD available today. Applications for ROM-PLD are numerous ranging from the replacement of ramdom combinatorial logic to the replacement of fast silicon PLDs. Because of the size of the ROM, it is possible to create reasonably large functions such as n-bit floating point adders and multipliers which are considerably faster than available single chip ECL floating point ICs. The speed of the 4K ROM is also fast enough to implement current and future FDDI (Fiber Distributed Data Interface) encoding/decoding. Other applications for the ROM are in forming in the areas of high speed control, mapping, code conversion, high speed sequencers, and state machines.

CONCLUSION

A 512 X 8, 4K ROM has been succesfully designed, characterized, and manufactured. An access time as short as 1.2 ns has been obtained. Extra margin in the circuit design for a wide range of power supply variation and processing window has contributed to good processing yield. The design criteria also allow the target pinch-off voltage to be changed for a particular application need in power and speed combination.

ACKNOWLEGEMENT

The authors wish to thank Bryant Welch, Yie-Der Shen, and Cathy Imboden for their efforts in the fabrication of the 4K ROM.

REFERENCES

1. Y. D. Shen, M.R. Wilson, M. McGuire, D. A. Nelson, and B. M. Welch, "An Ultra High Performance Manufacturable GaAs E/D Process", GaAs IC Symposium Technical Digest, pg. 125-128, 1987.

2. A. Fiedler, J. Chun, R. Edon, and D. Kang, "A GaAs 256 X 4 Static Self-Timed Random Access Memory", GaAs IC Symposium Technical Digest, pg. 89-92, 1986.

3. A. Fiedler, J. Chun, and D. Kang, "A 3 ns 1K X 4 Static, Self-Timed GaAs RAM", GaAs IC Symposium Technical Digest, 1988.

4. "All-GaAs Lineup Fuels High-Speed Digital Synthesizer" Microwaves & RF, pg 129-134, July 1988.

5. J. Chow, et al., "1.25 GHz 26-Pipelined Digital Accumulator". GaAs IC Symposium Technical Digest, 1988.

GaAs IC SYMPOSIUM

IEEE GALLIUM ARSENIDE INTEGRATED CIRCUIT SYMPOSIUM (10th, 1988, Nashville, Tenn.)
Sponsored by the IEEE Electron Devices Society
and cooperatively sponsored by the IEEE Microwave Theory and Techniques Society

10th ANNUAL

GaAs IC Symposium

TECHNICAL DIGEST 1988

⊙ NASHVILLE, TENNESSEE ⊙

NOVEMBER 6-9, 1988

GaAs IC Symposium
TECHNICAL DIGEST

Papers have been printed without editing as received from the authors.

All opinions expressed in the Digest are those of the authors and are not binding on the Institute of Electrical & Electronics Engineers, Inc.

Publication of a paper in this Digest is in no way intended to preclude publication of a fuller account of the paper elsewhere.

Copies of available volumes of this Digest may be obtained from The Institute of Electrical & Electronics Engineers, Inc., 445 Hoes Lane, Piscataway, N.J. 08854.

ii

A 3 NS 1K X 4 STATIC SELF-TIMED GaAs RAM

Alan Fiedler, Jino Chun, and Daniel Kang

GigaBit Logic, 1908 Oak Terrace Lane, Newbury Park, California 91320

ABSTRACT

A new GaAs 1K x 4 Static Self-Timed RAM (SSTRAM) with input and output latches and internal write-pulse generation has been designed, fabricated, and tested. Fully functional SSTRAMs have been obtained, with a worst-case clock access time (equal to read and write cycle time) of 3.6 ns for a 1.9 W device. This part is manufactured using 3 levels of interconnect metallization and GigaBit Logic's new High-Margin Enhancement - Depletion (HMED) process, and utilizes many innovative circuits.

INTRODUCTION

Today's supercomputer manufacturers are demanding memories of ever decreasing cycle times, combined with the flexibility afforded by input and output latches and internal write-pulse generation. To meet these requirements in a competitive industry, GigaBit Logic's standard depletion-mode MESFET process was enhanced with the addition of a lower pinch-off voltage device (for increased circuit design flexibility) and a third layer of interconnect metal (for increased array density). This enhanced process is combined with many design innovations to produce this manufacturable, high-performance SSTRAM. This memory uses GigaBit Logic's standard PicoLogic power supplies (Vss = -3.4 V and Vee = -5.2 V). In this paper, the SSTRAM's logic and timing diagrams are discussed, the new HMED process is outlined, and several of the circuits used, including the RAM cell, are presented. Finally, performance data is given.

ARCHITECTURE

Fig. 1 shows the RAM's block diagram and Fig. 2 a timing diagram for a read and write cycle. The operation of the RAM is straightforward. Both the read and write cycles begin on the falling edge of the clock, at which time the input latches enter their "open" state, and the output latches are latched with the previous output data.

On a read cycle (i.e., if the /WE input is high), four cells are selected, one for each quadrant, as determined by the input address. The selected cells then drive the data lines, which are in turn connected to the output latch. On the rising edge of the clock, the input latches enter their latched state and the output latches enter their transparent state, at which time the contents of the selected cells are passed through to the
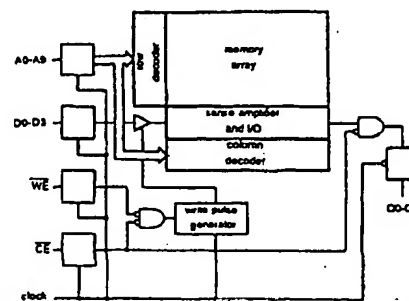

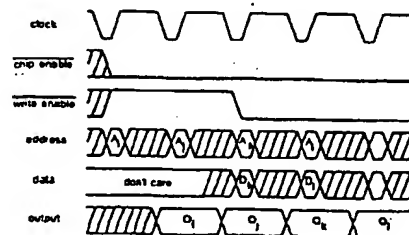Fig. 1. Architecture of 1Kx4 static self-timed RAM.

Fig. 2. Timing diagram of SSTRAM. Input latches are open when clock is low; output latches are open when clock is high.

output.

During a write cycle (i.e. if the /WE input is low), the input latches are transparent when the clock is low (as in the read cycle), while the ouput latches are latched. On the rising edge of the clock the input latches enter their latched state, and the output latches enter their transparent state. In write mode the contents of the input data latches then pass through to the_output. After a delay to ensure that the current address has selected the correct cells, the rising edge of the clock initiates a narrow write pulse which is applied to the selected cells. The timing and width of this pulse is internally V_p compensated, tracking the actual speed of the RAM. That is, a faster RAM (due to a higher depletion V_p) will inherently generate a narrower write pulse, delayed from the clock for a shorter period.

One redundant row and one redundant column per quadrant (four total) are included in the layout of this RAM to improve the yield. Repair is accomplished by laser-cutting third layer metal fuses.

PROCESS

This 4K SSTRAM is manufactured using GigaBit Logic's new High-Margin Enhancement - Depletion (HMED) process which combines two pinch-off voltages of high-performance, 1μm recessed gate MESFETs ($V_p = -0.65$ V and $V_p = -0.15$ V were selected for this work) for flexibility in low-power circuit design with 3 layers of metallization to reduce the cell size. This process uses 3" undoped LEC wafers. Si+ implantation through a very thin Si_3N_4 cap, 10X direct step on wafer photolithography, dry etching, and enhanced lift-off techniques. Using low implant energies, gate recessing, and rapid thermal annealing, high K values and transconductance are routinely obtained [1]. Typical DFET intrinsic K value and extrinsic transconductance (measured at $V_{gs}=0.6$-V) are 180 $\mu A/V^2\mu m$ and 250 mS/mm, respectively, while for $V_p = -0.15$ V "EFETs", they are 235 $\mu A/V^2\mu m$ and 240 mS/mm.

CIRCUIT DESIGN

Similar to previous work [2], for best speed-power product most of the RAM uses capacitor-FET logic, both in the form of common-source inverters and NOR gates and in the differential amplifiers used in the input latch and output latch / sense amp. The row and column decoders use diode-FET logic for the efficient layout of the required 6- and 4-input NOR gates. All word lines in the array, row and column address lines, as well as data and control lines, are driven using the super-buffer shown in Fig. 3a. The advantage of this buffer is that the second stage consumes no DC power by using an enhancement pull-up FET and series diode. This "enhancement" pull-up FET in the second stage uses

the same V_p of -.15 V as the enhancement FET in the cell and elsewhere in the RAM. This reduction in power is achieved with the additional benefit of a 600 mV reduction in output voltage swing, thereby decreasing gate delay, as compared to the super-buffer shown in Fig. 3b, whose output voltage swing is larger than required for complete switching of the next gate.

The capacitor used in this logic is not a reverse-biased Schottky diode, as has been used in the past [2], but rather a metal-insulator-semiconductor (MIS) capacitor, where the metal is first-layer metal, the insulator is a thin layer of Silicon Nitride, and the semiconductor is GaAs implanted with all three implants (N'd, N'e, and N+). At 2 V reverse bias, the diode capacitance is measured to be 1.43 fF/μm^2 while the MIS capacitance is measured to be 1.24 fF/μm^2 and is independent of bias. At the expense of this slight reduction in capacitance per unit area, we reduce typical room temperature leakage currents across the capacitor from 10 nA/μm^2 (for the Schottky diode) to .0004 nA/μm^2 (for the MIS capacitor). This allows a dramatic reduction in level-shift current, and a considerable savings in power.
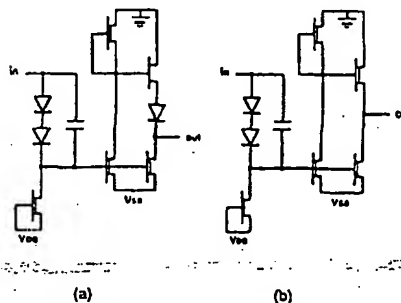

(a)   (b)

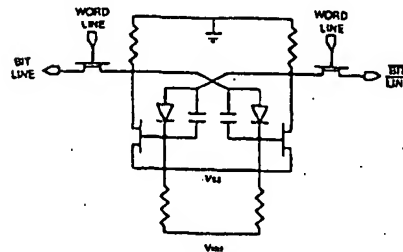Fig. 3. (a) Low power super-buffer (b) Standard super-buffer.
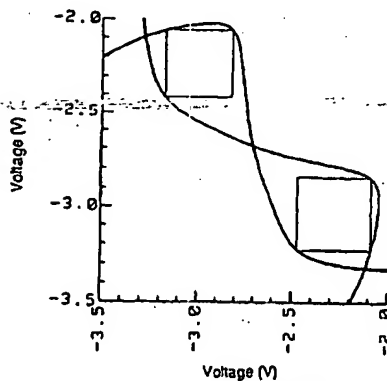

Fig. 4. HMED RAM cell.

The memory element in this work is the HMED cell shown in Fig. 4. This cell achieves excellent stability and bit-line drive capability by using a single-diode level-shift from the output of each inverter to the gate of the other. This allows for a more negative pinch-off (V_p of -.15 V is used) switching FET to increase bit-line current drive while still improving stability and manufacturability over a traditional E/D cell using DCFL. MIS capacitors in parallel with the level-shift diodes make the write operation (by forcing a bit line

low) fast. The use of MIS capacitors, rather than reverse-biased diodes as used in our 12G014 1K RAM, should give reduced photocurrent collection for better and single-event-upset immunity. The cell takes advantage of these low-leakage MIS capacitors by using high-value Cermet resistors to bias the level-shift diodes and as loads for the latch inverters. Fig. 5 shows the cell transfer curves for a deselected and selected cell. These show good noise margins of 480 mV and 350 mV, respectively. Cell size has been reduced 20% (as compared to GigaBit Logic's 2-level metal process) by using 3 layers of metal, with cell interconnect on first and second layer metal, the bit lines on second layer metal, and the word line on third layer metal. Also, the cermet resistors lie over the MIS capacitors, thereby using no additional layout area. Cell size is 40.2 μm x 35.4 μm.

The output driver is configurable as a low-impedance driver for a parallel, or "far-end", terminated (to -2 V) 50 Ω transmission line or as a 50 Ω series, or "back", terminated driver for an unterminated 50 Ω transmission line (Fig. 6). In the series terminated case, the output is DOST, and the pads DOPT and DOCS are shorted. For the parallel terminated case,
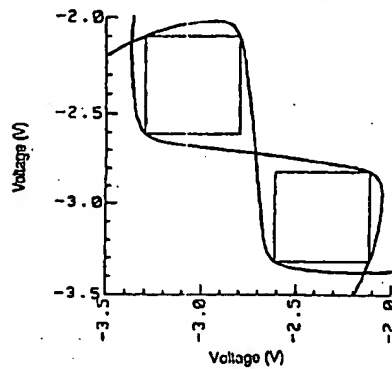
(a) Deselected cell, showing 480 mV noise margin.

(b) Selected cell, showing 350 mV noise margin.

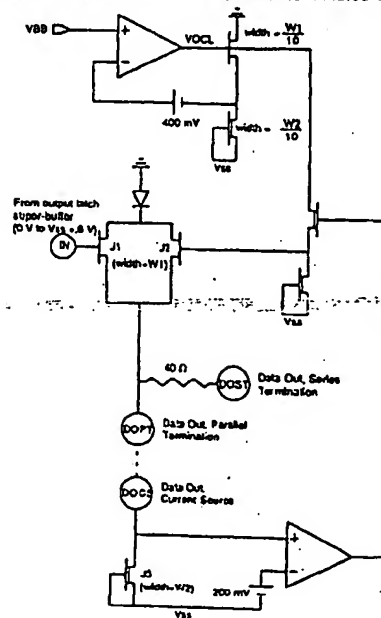Fig. 5. HMED RAM cell transfer curves at 25 C.

Fig. 6. Output driver, configurable as a parallel- or series-terminated driver.

the output is DOPT, and the pads DOST and DOCS are open.

For this series-terminated case, it is important to achieve 50 Ω output resistance in addition to the correct ECL output levels. The current through J3 biases J1 or J2 so that the resistance looking into the sources of J1 and J2 in parallel is always about 10 Ω. This 10 Ω resistance adds to an on-chip 40 Ω resistor to achieve the required 50 Ω output resistance. An output high voltage of about -800 mV is achieved by level-shifting the drain voltage in the output FETs by 1 diode from ground. To achieve an output low voltage 400 mV below VBB, an on-chip circuit generates the voltage VOCL (voltage output clamp low) which is applied to the gate of the output clamp FET, J2, by the action of A1, since DOCS can never drop below Vss + 200 mV. When the gate of J1 is low, all of the bias current into J3 flows through J2 and the gate voltage on J2 (VOCL) is such that the output low voltage will be 400 mV below VBB (approx. -1.7 V).

For the parallel-terminated case, DOCS is open, and the output clamp FET, J2, is disabled by the action of the amplifier A1. The output signal then swings between -800 mV and V_H (-2 V), since the gate of J1 is switching between 0 and Vss + 700 mV (-2.7 V).

TEST RESULTS

Fully functional 4K RAMs have been obtained, and the yield of repairable 4K RAMs has recently been excellent. No pattern sensitivity is seen, and the RAMs routinely pass checkerboard, march and galloping patterns. The ratio of repairable RAMs to fully functional RAMs has been very high, lending support to the decision to include redundancy in this RAM. Fig. 7 shows the clock access time of a 1.86 W part, as well as some internal waveforms.
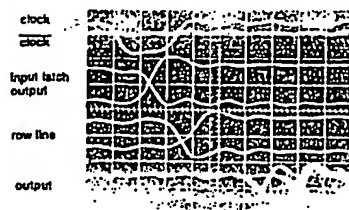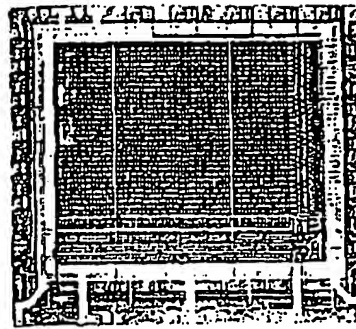
Fig. 8. Photograph of completed 1K x 4 SSTRAM. Die size is 4280 μm x 3905 μm.

CONCLUSION

A high-speed, self-timed 1K x 4 static RAM with input and output latches has been designed, fabricated, and characterized. Standard power-supply voltages and ECL I/O levels make this RAM compatible with existing high-speed ECL systems. By latching input and output signals and generating the write pulse internally, this RAM can be an important and useful component of high-speed cache memory.

ACKNOWLEGEMENTS

Gratitude is extended to GigaBit's fab engineers Yie-Dor Shen, Mickey McGuire, Mark Wilson, and Bryant Welch for their efforts in the development of the HMED and 3-level metal process, and to Tom Coleman for his work on RAM characterization.

Fig. 7. Waveforms (measured at wafer-probe) associated with the read cycle of a 1.86 W RAM. True vertical scale is 2 V/division. Output waveform is the super-position of every transition of a row-last and column-last checkerboard pattern, and shows a worst-cast clock access time of 3.6 ns. The column access time is the faster access time.

REFERENCES

1. Y. D. Shen, M. R. Wilson, M. McGuire, D. A. Nelson, and B. M. Welch, "An Ultra High Performance Manufacturable GaAs E/D Process". GaAs IC Symposium Technical Digest, pp. 125-128, 1987.

2. A. Fiedler, J. Chun, R. Eden, and D. Kang, " A GaAs 256 X 4 Static Self-Timed Random Access Memory". GaAs IC Symposium Technical Digest, pp. 89-92, 1986.

GaAs IC SYMPOSIUM

IEEE GALLIUM ARSENIDE INTEGRATED CIRCUIT SYMPOSIUM
Sponsored by the IEEE Electron Devices Society
and cooperatively sponsored by the IEEE Microwave Theory and Techniques Society

10th ANNUAL
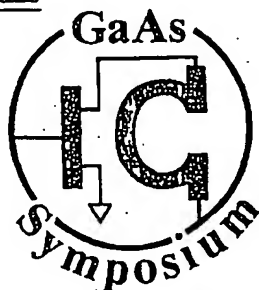
GaAs IC Symposium

TECHNICAL DIGEST 1988

⊙ NASHVILLE, TENNESSEE ⊙

NOVEMBER 6-9, 1988

GaAs IC Symposium
TECHNICAL DIGEST

TK
7874
113
1988

Papers have been printed without editing as received from the authors.

All opinions expressed in the Digest are those of the authors and are not binding on the Institute of Electrical & Electronics Engineers, Inc.

Publication of a paper in this Digest is in no way intended to preclude publication of a fuller account of the paper elsewhere.

Copies of available volumes of this Digest may be obtained from The Institute of Electrical & Electronics Engineers, Inc., 445 Hoes Lane, Piscataway, N.J. 08854.

A DATA-TRANSFER ARCHITECTURE
FOR FAST MULTI-BIT SERIAL ACCESS MODE DRAM

A. Takasugi, Y. Ohtsuki, A. Kamo and M. Uesugi

Oki Electric Industry Co.,Ltd., Tokyo Japan

ABSTRACT
A data-transfer architecture for both fast multi-bit serial ac-
cess mode and multi-output pin configuration DRAM is described in
this paper. The key feature of the newly developed architecture
to realize fast serial access time is the concurrent data-trans-
fer of two cascade serial outputs in a $\overline{CAS}$ cycle using time-mul-
tiplexed data-bus. The data-transfer per one output pin is
achieved by only two pairs of time-multiplexed data-bus. The data
bus enables to minimize die size compared with non time-multi-
plexed data-bus; conventional technique.
By using the architecture, a 64K x 4b nibble mode DRAM of small
die size and fast nibble access time has been developed.

INTRODUCTION
Microcomputers are absorbing a greater share of total DRAM
market. Microcomputer hardware requires DRAMs having performances
such as small size, fast access time and/or simpler design, less
granularity and wider band width. [1],[2],[3]
For those requirements, new DRAMs having both multi-bit serial
access and multi-bit parallel data-out organization are required.
Conventional technique, however, requires large number of data
bus and other circuits to be connected to each data bus for the
organization. They require large area on a die and force to widen
the die size. A large die has disadvantage in respect of cost,
packaging and reliability.
In order to reduce the die size, novel data-transfer architec-
ture has been devised. The architecture also assures the same
serial access time with that of conventional nibble mode.

CIRCUIT OF THE ARCHITECTURE
M-bit serial access mode and N output-pin configuration DRAM
designed by conventional data-transfer architecture requires MxN
pairs of data bus and MxN latches. In addition to that, large
number of interconnections between data bus and other circuits
are required. They are the large factors widening the die size.
In order to solve those problems, a novel data-transfer archi-
tecture reducing the number of data bus and other circuits has
been devised. The architecture requires only 2N pairs of data
bus, 2N latches.The architecture enables large die size reduction
despite the additional M-2 lines of control signals and control
circuits.
The circuit of the architecture per one I/O buffer is shown in
FIGURE 1. The circuit consists of two blocks; the block-A and
the block-B. In this circuit, 2n bits of data are accessed
serially.

ϕ_{A1}-ϕ_{An} and ϕ_{B1}-ϕ_{Bn} are on/off control signals between data bus pairs and bit line pairs. In the block-A and in the block-B, the data bus DB-A and DB-B are time-multiplexed, respectively.

DATA-TRANSFER OF THE ARCHITECTURE

(a)-(c) in FIGURE 2 show the data-transfer of the architecture using the simplified block diagram of the circuit in FIGURE 1. In FIGURE 2, the first output data is designated to the data DA1, and then the data DB1,DA2,DB2...,DAn,DBn are accessed serially.

Actually, each data out of the 2n data in FIGURE 2 is to be selected as an first data according to the input address. After that, succeeding serial data are accessed. In the followings, the data-transfer architecture is explained according to the CAS toggle cycle.

(a) In the cycle a, the data DA1 and DB1 are transfered simultaneously from memory cells to the latch L1 and L2 located near the I/O buffer, respectively. And the data DA1 is transfered to the I/O buffer and then driven out. On the other hand, the data DB1 is latched by the latch L2.

(b) In the cycle b, the block-A is initialized. On the other hand, the block-B maintains the state in the cycle a; the data DB1 is latched for the next fast access without being reset.

(c) In the cycle c, the data DB1 latched by the latch L2 in the cycle a is transferd to the I/O buffer and then driven out. Fast access time is enabled because the data is latched in the previous cycle; the data-transfer time from memory cell to latch is not required. On the other hand, the data DA2 is transfered from memory cell to the latch L1. This latched data enables fast access in the next cycle.

Succeeding data are serially accessed in the same data-transfer operations as those in (b),(c). The concurrent data-transfers of two cascade serial output data in one CAS cycle as shown in (c) enables fast serial access time.

High speed serial write is also achieved in almost the same way as explained in FIGURE 2. The differences are the latch reset timing and the data-transfer direction.

SIGNAL TIMINGS OF THE ARCHITECTURE

FIGURE 3 shows the clock timings of the architecture in the circuit explained in FIGURE 1. The ϕ_{A1}-ϕ_{An} and ϕ_{B1}-ϕ_{Bn} are main signals for multiplexing the data bus DB-A and DB-B, respectively. FIGURE 4 presents the block diagram of the architecture. The signals multiplexing data-bus for fast serial access are controled by a 2n-bit shift register.

The 2n-bit shift register consists of 2n master/slave flip-flops as shown in FIGURE 5. The control signals of the architecture are generated by the outputs of each master and slave flip-flops.

FIGURE 6 shows the simplified logics of the control signals in serial access mode. The signals of MAi and SAi series control the main signals to multiplex the data bus DB-A, and the signals of MBi and SBi series control the main signals to multiplex the data bus DB-B.

In the first CAS active cycle (normal cycle), the two successive on/off control signals between bit line pairs and data bus pairs are selected for the data-transfer of first output data and next output data by the input address as shown in FIGURE 7. The signal for the next output enables the fast serial access

in the next $\overline{CAS}$ active cycle.

APPLICATION OF THE ARCHITECTURE

A 64K x 4b nibble mode DRAM has been developed using the architecture. FIGURE 8 shows the die photograph of the DRAM. The architecture has ensured small die size comparable with standard 256K DRAM and fast nibble access time of 20 nsec. Especially the reduction of data buses from 16 pairs to 8 pairs has enabled packaging in a 18 pin standard 300 mil plastic DIP. The slim die has been achieved without any modifications of memory cell array design that is currently applied to standard 256K X 1b DRAM. FIGURE 9 shows the waveforms of the DRAM. Table 1 shows the characteristics of the DRAM.

CONCLUSIONS

A novel data-transfer architecture for DRAM has been proposed. The architecture is effective in reducing die size of both multi-bit fast serial access mode and multi-output pin configuration DRAM.

The key feature is the concurrent data-transfer of two cascade output data using time-multiplexed data-bus. The time-multiplexed data-transfer operations are controled by a shift register which consists of master/slave flip-flops.

A 64K x 4b nibble mode DRAM designed by the architecture has been developed. The architecture has enabled die size reduction and fast nibble access time of 20nsec. Especially the width size reduction has enabled a slim die comparable with that of standard 256K DRAM. The slim die is suitable for packaging in a 18 pin standard 300mil plastic DIP.

ACKNOWLEDGMENTS

The authors would like to thank T.Kobayashi and M.Ino for their continuous encouragement, and A.Takakura, Y.Iwata, and T.Higashi for their constant support.

REFERENCES

[1] K.Fujishima, et all., "A 256K Dynamic DRAM with Page-Nibble Mode", IEEE VOL. SC-18, No.5 P.470-478 ; 1984.
[2] F.Baba, et all., "A 64K DRAM with 35 ns Static Column Operation", IEEE SC-18, No.5, P.447-451 ; 1984
[3] S.Suzuki, et all., "A 128K word X 8 b DRAM", ISSCC'84 TECHNICAL DIGEST, P.106-107 ; 1984.
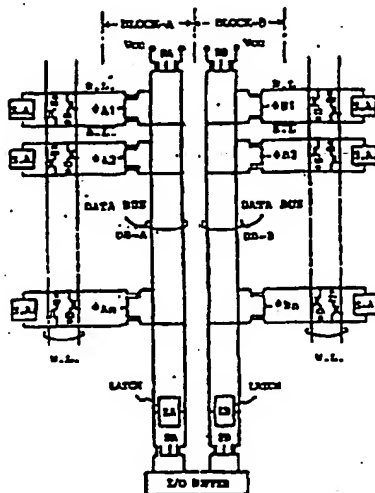
FIGURE 1 - Circuit of the architecture

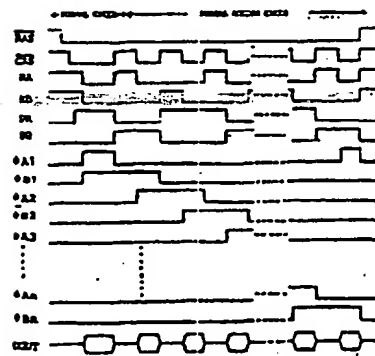FIGURE 2 - Nibble mode operation using the new data-transfer architecture.

FIGURE 3 - Main signal timings of the new data-transfer architecture.
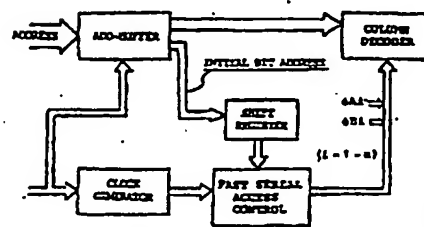
FIGURE 4 - Block diagram of the fast serial access control.
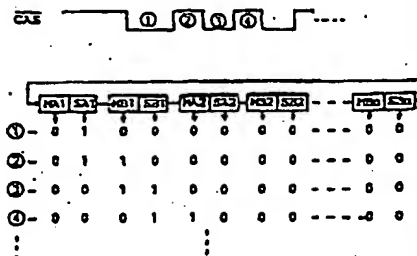
FIGURE 5 - Shift register operation.
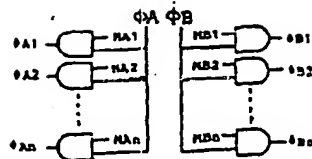
FIGURE 6 - The logics of control signals.
(serial access cycle)
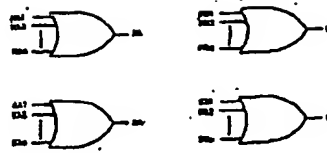
FIGURE 7 - Selection of control signals.
(normal cycle)

FIGURE 9 - The waveforms of the DRAM.
(Vcc=5V , Ta=25°C)

FIGURE 8 - The microphotograph of
64K x 4b DRAM.

* ORGANIZATION   64K WORDS X 4 BITS
* MODE   NIBBLE MODE / PAGE MODE
                        (METAL MASK OPTION)
* RAS ACCESS TIME   100 nsec
* CAS ACCESS TIME   50 nsec
* NIBBLE ACCESS TIME   20 nsec
* ACTIVE CURRENT   70 mA   TRC=200nsec
* STANDBY CURRENT   4 mA
* REFRESH CYCLE   256 CYCLES/ 4msec
* DIE SIZE   4.00mm X 9.50mm
* CELL SIZE   5.50um X 12.25um
* PROCESS   2um DESIGN RULE
                        DOUBLE LEVEL POLY
                        DOUBLE LEVEL METAL

TABLE 1 - The characteristics
of the DRAM.

Burst Mode Memories Improve Cache Design

Zvie Amitai, Product Planning and Applications Manager
David C. Wyland, Vice President of Engineering
Quality Semiconductor, Inc.
851 Martin Avenue
Santa Clara, CA 95050-2903

Tel: (408) 988-8326   Fax: (408) 496-0591

ABSTRACT

Burst mode memories improve cache design by improving refill time on cache misses. Burst mode RAMs allow refill of a four word cache line in five clock cycles at 50 mHz rather than the eight clock cycles that would be required for a conventional SRAM. Burst mode RAMs also have clock synchronous interfaces which make them easier to design into systems, particularly at clock rates of 25 mHz and above.

BURST MODE SRAMS

A burst mode RAM provides high speed transfer of a block of sequential words, called a burst. A block diagram of a burst mode SRAM is shown in Figure 1. A burst mode RAM consists of a conventional SRAM plus an address counter, a read/write flip flop and a write register. Read and write timing is contro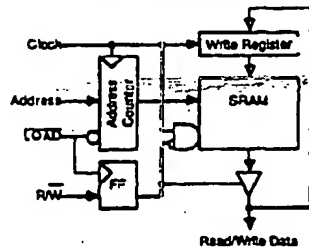lled by a clock in combination with the address counter load and read/write signals. In this configuration, random access to a word in the SRAM requires two clock cycles with successive words being read or written at one clock cycle per word. This is shown in the timing diagrams of Figures 2 and 3.

Figure 1: Burst RAM Block Diagram

In the read timing diagram of Figure 2, the first clock cycle is used to load the address counter and the read/write flip flop for random access to the first word. Read data comes out of the SRAM before the end of the second clock cycle. The address counter is incremented at the end of the second clock cycle, and the next word is read from the SRAM. This allows one clock cycle per successive word read following the initial random access.

Figure 2: Burst RAM Read Timing
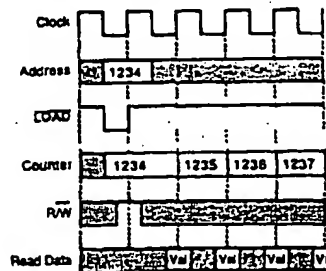
For write operations, the first word of data to be written is clocked in to the write register at the same time the address counter and the read/write flip flop are loaded, as shown in Figure 3. Data from the write register is written into the SRAM during the second clock cycle. At the end of the second clock cycle, new data is clocked into the write register and address counter is incremented to the next location to write the next sequential word.

Figure 3: Burst RAM Write Timing

279
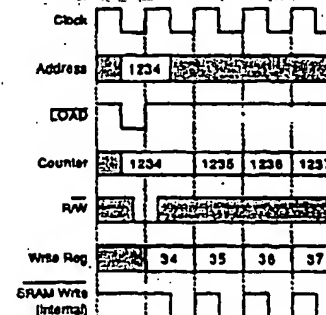
The burst mode memory is capable of high speed operation after the initial access because the sequential addresses are generated internally by the address counter. This greatly reduces the read and write cycle times for sequential data following the first access. Clock speeds of up to 50 mHz are possible in a TTL system, making the burst mode memory particularly well suited to the newer generations of high speed RISC and CISC chips.

Burst mode RAMs are faster than SRAM based memory systems because the address counter is integrated into their design. In a burst mode SRAM, the minimum cycle time of the burst operation is approximately the same as the address access time of an equivalent SRAM. This can be as low as 20 ns. In a conventional burst mode memory system design using an SRAM and an address counter, the minimum minimum cycle time is determined by the sum of the clock to output delay of the counter plus the address access time of the SRAM. The cycle time is therefore increased by the delay of the address counter. This adds 6.2 ns to the memory cycle time using the QSFCT161A, one of the fastest counters commercially available. If a 20 ns SRAM is used, the minimum cycle is 26.2 ns. Alternately, a 13.8 ns SRAM would be required to achieve the 20 ns cycle time of a burst mode RAM.

CACHE MEMORY IN RISC AND CISC PROCESSORS

The use of cache memories has become a standard feature of high performance processor design. Indeed, RISC design is based on cache memory. The function of a cache memory is to improve the effective access time of the main memory, usually medium speed DRAM, by eliminating processor wait states. The cache does this by keeping copies of the most frequently read words from main memory in a small, high speed buffer memory. When the processor attempts to read a word from main memory, the cache checks to see if it has a copy. If it does, it responds immediately. If not, the main memory is accessed on a normal read cycle, and the processor waits for it to respond. The cache therefore speeds up the system by reducing the average amount of time the processor has to wait to read a word from memory.

Caches are effective because most of the memory accesses are read cycles from a relatively small cluster of memory locations, in typical programs.

Cache performance can be defined in terms of effective wait states with a cache relative to the number of wait states without it. A 33 mHz processor with medium speed DRAM memory may require three wait states without a cache and 0.5 wait states with a cache. The three wait states without a cache are determined by the timing requirements of the main memory. The 0.5 wait states is a statistical average. It can be estimated by the product of cache miss rate and the number of wait states required for cache refill on a miss.

A direct mapped cache for a 32-bit processor is shown in Figure 4. A direct mapped cache consists of a cache tag RAM, a cache data RAM and a small amount of logic to control events when a cache hit or a cache miss occurs. A cache hit is said to occur if a requested word is found in the cache. A miss occurs when the word is not found in the cache.

Figure 4: Cache Block Diagram
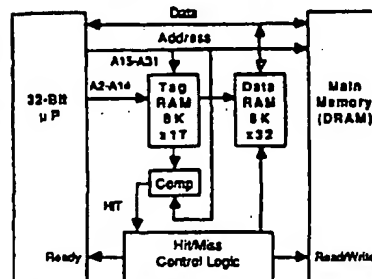
The cache stores copies of words read from main memory in the cache data RAM and stores the location these words are read from in the cache tag RAM. In the direct mapped cache, the least significant bits of the address bus are sent to both the tag and data RAMs while the most significant bits are stored in the tag RAM when data is stored in the cache data RAM. In the example shown, both the tag and data RAMs are 8K words deep.

When a read request is made to main memory, the least significant bits of the address are used to select one of the 8K words in both memories. The most significant bits of the address are compared against the bits stored in the tag RAM. If there is a match between the two, then the data stored in the data RAM is a copy of the data at the requested location and can be immediately supplied to the processor. This is a cache hit. If the upper address bits do not match, the data stored came from a different location. This is a cache miss.

Direct mapped caches work because most accesses to main memory are typically to a small cluster of a few thousand words located somewhere in the memory space. If the cache is larger than this cluster size, most of the read data will be provided by the cache. The least significant bits of the address bus are used to index within this cluster of words, and the most significant bits identify the region of memory that they came from. (Cache theory is a little more subtle than this. It treats the least significant bits of the address as a hashing function for a hash indexed buffer.)

280

Figure 6: 80486 32K Byte Cache Block Diagram
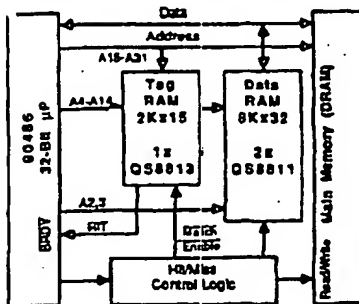
The design of Figure 6 uses one QS8813 8Kx18 Tag RAM and two QS8811 8Kx18 Burst Mode RAMs for the tag and data memories respectively. The QS8813 is an 8Kx18 Tag SRAM with built-in match enable logic that allows it to directly drive the BRDY input of the 80486. This eliminates the need for additional logic in the propagation delay path between the Tag SRAM and the microprocessor. This can save five or more nanoseconds in match time. Only 2K of the 8K are used; however, the 8813 provides a single chip design solution for the TAG RAM. The complete design requires only three RAM chips.

Figure 7: 80486 128K Byte Cache Block Diagram
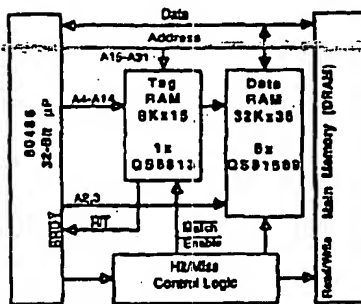
Figure 8: 80486 Cache Timing Diagram
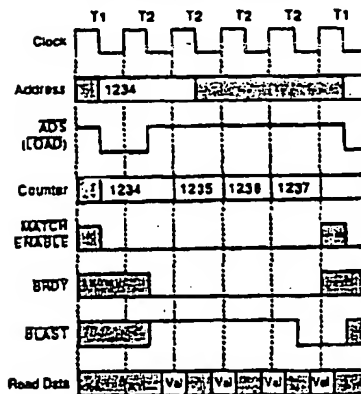
The design of Figure 7 uses one QS8813 8Kx18 Tag RAM and four QS8339 32Kx9 Burst Mode RAMs for the tag and data memories respectively. The full 8K words of the 8813 are used to support the 32K words of the 8339.

Both the 8811 and 8339 Burst Mode RAM chips provide an on-chip address counter and logic for burst mode operation. The address counter provides for bursts of up to four words using the 80486 address counting algorithm. Also, the burst counter on the 8811 count in either binary or 80486 counting modes, pin selectable.

CONCLUSION

Burst mode memories provide performance improvement for the cache systems used in high speed CISC and RISC systems which use multiple words per cache line. They are particularly useful at CPU clock speeds above 25 mHz due to their higher performance and simpler interface. Because of these advantages, burst mode memories are becoming a standard component for cache design of high speed systems.

281

72

100 MHz Serial Access Architecture for 4Mb Field Memory

Hiroaki Ikeda, Akira Tsujimoto, Yoshinori Sato*, Junji Tajima,
Takeo Adachi*, Kunihiko Hamaguchi*, Naohiro Fukuhara* and Mayo Miyauchi
LSI Memory Division, NEC Corporation
*First Memory Engineering Department, NMS Corporation
1 - 35, Miramihashimoto 3 - chome, Sagamihara Kanagawa 229, Japan
*403 - 53, Kosugimachi 1 - chome, Nakahara-ku, Kawasaki 211, Japan

1. INTRODUCTION

For advanced TV and VCR equipment, particularly those for HDTV (High Definition TV) equipment, high-speed, high density serial access memory or field memory is required for storing the video data for each field. So far several such devices have been reported[1], but have such shortcomings as large circuit area for serial I/O circuits and complex controls. Especially, high speed redundancy circuit architecture to repair damaged memory cells is not cleared.

In this paper, we will present a new architecture that improve serial I/O operation speed, reduce layout area and enable simple control significantly. We have employed an architecture featuring high-speed, simple configuration and easy controllable data shifter and high speed redundancy circuit. With this architecture, we have developed a 4Mb field memory of 100MHz serial access capability. The process technology used is a 1.0μm CMOS, and the die size is 12.94mm x 25.9mm.

2. CONFIGURATION OF 100 MHz HIGH DENSITY FIELD BUFFER

The configuration of the 100MHz 4Mb field memory is shown in Figure 1. The field memory has a 568 lines x 960 pixels x 8 bit (4362240-bit) memory cell array designed for HDTV screens. The detailed configuration of the memory cell array division is shown in Figure 2. The memory cell array has been divided into four blocks A, B, C, and D. Each block has been provided with a data register which reads and writes serial I/O data simultaneously since the field memory performs read and write operations in the same cycle.

An explanation of the transfer sequence is given below. Suppose that a serial I/O data operation is performed by block B. Simultaneously, data is transferred from the memory cells to the C block data register, which is the next block to perform I/O operations. The data entered into the A block data register is transferred to the memory cells before block B I/O operations are completed. Thus the blocks are serially accessed in the order A, B, C and D.

The data register can be configured in two ways. One is the address pointer shift method which moves an address pointer to the register (typically this method has been used in a Dual Port Graphics Buffer (VRAM)[2]). The other is the data shift method which transfers the I/O data to the shift register. A high speed and high density field memory requires a high operating speed to cope with the increasing writing impedance caused by high density circuit design.

For that reason, the data shift method has been chosen as the serial access architecture of the 4Mb field memory.

3. DATA SHIFT METHOD

The data shifter used in the field memory is shown as the circuit diagram in Figure 3. It consists of shift register and transfer gates. The employment of the data shift method has a number of advantages. First, I/O data is shifted in the shift register synchronously with the clock signal, data can be input and output with the use of the CLK* and CLK* signals only. Consequently, shift operations are independent on the number of unit cells in the register, so operations can be carried out at the high speed corresponding to clock frequency. Secondly, unlike the pointer shift method, there is no need for a write / read bus and a write / read address pointer in memory array, which means that the chips can be made smaller. As the read and write operations of the field memory are performed simultaneously and synchronously, the same shift register can be used for both read and write operations. So, it has been possible to employ the data shift method to make further reductions to chip size.

4. REDUNDANCY

The description of the type of redundancy used in the data shift method is shown in Figure 4. Since high speed operation is top priority of the field memory, column redundancy operations also have to be performed at high speed. The following describes the operation of column redundancy. In Figure 4, the fuse corresponding to the number of column redundancies used is cut off. This causes the first data entered into the redundancy write shifter is led to the first address of the redundancy read shifter and stay there. Increases reading speed and simplifies control. As a result, the speed of the 1.0 μm CMOS field memory has been raised to 100 MHz while space has been saved by eliminating the need for bus and pointer.

5. 4MB FIELD MEMORY CHIP

As shown in Figure 5, the chip consist of two identical 2Mb memory portions, which are connected each other internally. A dynamic test result of the memory chip is shown in Photograph 1, where access operations were performed in a clock cycle of 9ns. It can be seen that 100MHz operation is adequately satisfied.

6. CONCLUSION

The use of the data shift method and high speed column redundancy circuit enabled high speed data I/O operations, simultaneous read and write operations, chip size reduction and the sharing of registers. The adoption of these circuits led to the realization of a 4362240-bit field memory. As a result, the data register circuit area of the prototype chip was 40% smaller than that of

conventional chips. It has been confirmed that the frequency of serial read and write operations is 100 MHz.

7. ACKNOWLEDGEMENT

The authors wish to thank J.Takashima, H.Yamamoto and S.Ariizake for their continuous encouragement throughout this work.

8. REFERENCES

(1) H. Kotani et. al. "A 50 MHz 8 Mb Video RAM with a Column Direction Drive Sense Amplifier" Symposium on VLSI Circuits Digest of Technical Papers, pp 105 - 106, 1989.
(2) S.Ishimoto et. al. "A 256K Dual Port Memory", ISSCC Digest of Technical Papers, pp 38 - 39, 1985.

Figure 3 Circuit Diagram of Shift Register
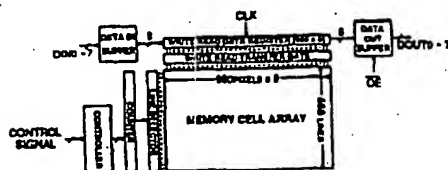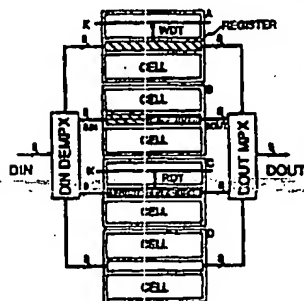

Figure 1 Block Diagram of the Field Memory
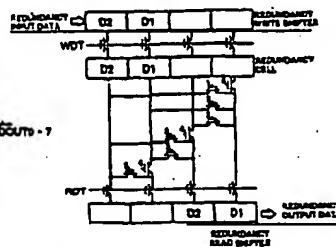
Figure 4 Column Redundancy

Figure 2 Physical Array Division

Figure 5 Chip Layout

CLK

DOUT

Photograph 1 Typical Operating Waveform of the Field Memory (5ns / div.)

12
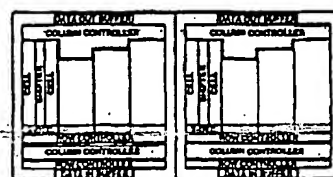
Pipeline Architecture for Fast CMOS Buffer RAM's

DORIS SCHMITT-LANDSIEDEL, BERNHARD HOPPE, GERD NEUENDORF,
MARIA WURM, AND JOSEF WINNERL

Abstract —This paper describes a new pipeline architecture for CMOS SRAM's that allows operation at very high clock rates. Basic requirements for achieving the high speed are the implementation of a hierarchical architecture and a memory cell with separate READ and WRITE data lines. Experimental results of the access speed of hierarchically organized memory blocks are between 1.5 and 3.5 ns. The maximum operating frequency of a 16K pipelined hierarchical SRAM (PHSRAM) is in the range of 300 MHz.

I. INTRODUCTION

SYNCHRONOUS SRAM's, also called registered SRAM's, that contain input and output latches [1], [2] are used in clocked systems to avoid additional external synchronization circuitry. Thus the maximum clock frequency can be increased with respect to asynchronous devices. Memories with a further degree of pipelining can be applied when a high data throughput is important, whereas a delay of several clock periods between address input and data output can be tolerated. An obvious domain of application is in digital signal processing. If, for example, variable filter coefficients have to be provided with high clock frequency or data have to be stored intermediately. Other uses could be in testing equipment for high-speed signal generation and intermediate storage of measurement data as well as in video and graphic systems.

Synchronous SRAM's can be used for these purposes. Their maximum operating frequency is determined by the access time of the complete on-chip memory, except the I/O circuits. According to the pipeline strategy, the clock frequency can be increased by introducing more pipeline stages. In conventional memory architectures, however, it is hardly possible to split up the critical path between the

I/O circuitry into several pipeline stages. Registers might be inserted after the address predecoder, but this section consumes only a small portion of the access time. The predecoded address signals branch into a large number of word-line signals, where it would be impractical to insert further registers. Due to the reduced signal level on the bit lines, the data signals can be stored in a digital register only after the sense amplifier. The largest portion of the access time is consumed by word-line delay and sensing, where no register stage can be inserted. Thus only a small increase in operating frequency could be obtained by additional pipelining in synchronous SRAM's.

In this paper, we present a new approach for the design of fast pipelined buffer SRAM's [3]. To circumvent the above-mentioned limitations for further pipelining of a memory, we introduce a highly hierarchical memory structure as proposed in [4] and [5]. More additional registers can be inserted between the different levels of hierarchy to subdivide the critical data path of the RAM into several pipeline stages. Therefore, the pipelined hierarchical RAM (PHSRAM) can be operated at a clock frequency several times higher than a conventional synchronous RAM.

A seven-transistor memory cell with separate word and data lines for READ and WRITE operations is a key component for the effective realization of a PHSRAM. Using this cell, no WRITE recovery time is required for bit-line equilibration as is the case for conventional static RAM cells, and READ or WRITE cycles can be performed with the same clock frequency. The cell provides a full logic swing and no analog sensing circuitry is required. To reduce the power consumption of the PHSRAM we used a selective clocking scheme, where only the relevant subblocks and registers are activated.

To demonstrate the speed achievement of this architecture, we realized several subblocks of hierarchical memories in a 1-μm CMOS technology. Because the lowest order subblock is the time-critical stage of the pipeline, the delay time of one subblock plus a register delay represents the minimum clock period of the PHSRAM. Measurement results are given in Section VII.
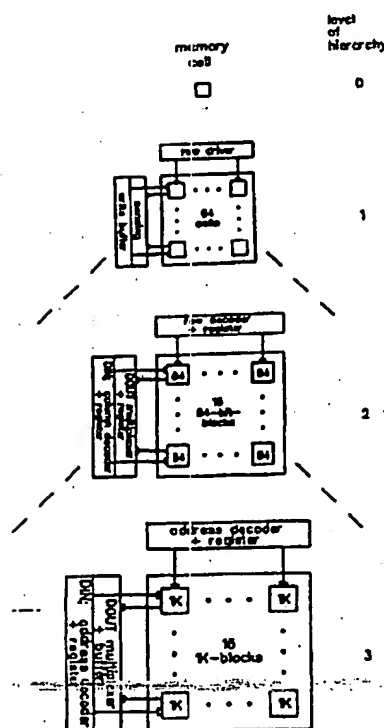
Fig. 1. Structure of a 16K SRAM with hierarchical architecture.

II. ARCHITECTURE OF THE PHSRAM

Fig. 1 illustrates how a 16K SRAM can be partitioned into four levels of hierarchy. In the terms of Rem and Mead [4], the elements of level 0 are the memory cells themselves. As the first-level elements, 64-b blocks are chosen consisting of 8×8 cells and peripheral circuits for select-line drivers and sensing. The elements of the second level are 1K blocks consisting of 4×4 64-b blocks and peripheral circuits. The third level is the total 16K block.

In the hierarchical architecture the critical path delay is distributed over the hierarchical levels and does not occur mainly on the word and bit lines as in conventional SRAM's. Therefore, it is possible to divide the critical path into pipeline stages by inserting registers at the interfaces of the topmost hierarchical levels, i.e., levels 2 and 3 in our example. No pipeline stages are inserted on the lowest hierarchical levels, as this would require too many registers.

Maximum operating frequency is obtained if the delay in the most time-critical stage is as small as possible. Therefore, we deviate from Rem and Mead's original proposals. The memory blocks in the lower hierarchical levels are not supplied with complete decoding and data multiplexing circuitry. Low-level address decoding and data multiplexing is partly done in the higher levels of hierarchy. For instance, each 1K block (level 2) generates four data-out signals. The relevant data line is selected in level 3. Or, as a second example, activation of the rows and columns of memory cells of the 64-b blocks (level 1) is triggered by row- and column-select signals, which are generated in predecoders on levels 3 and 2.

Fig. 2 shows the resulting architecture for the 16K PHSRAM with five pipeline stages. In the first pipeline stage, the four highest addresses are used to create 16 block-select signals for the 1K blocks. The lower addresses are predecoded into four groups of select signals: row and column-select-2 signals are destined to select the rows and columns of 64 blocks within a 1K block, whereas row and column-select-1 signals address the cells within a 64-b block. The input data DIN and the write-enable signal WE are stored in synchronizing registers.

In the second pipeline stage, the predecoded signals are distributed to the 1K blocks. They are latched in registers connected to block selective clock signals (see Section III). In the third pipeline stage, data are written to or read from the 1K block. In the example, four data-out latches are provided in the 1K block. In the fourth and fifth stages, the output data are further multiplexed and buffered.

III. PIPELINE OPERATION AND SELECTIVE CLOCKING

A timing diagram of the data flow through the pipeline stages is shown in Fig. 3. A READ cycle is followed by a WRITE cycle. The READ address bits are latched in cycle 1 (falling edge of clock PM), the corresponding memory cell is addressed during cycle 4, and the data bit appears at the output during cycle 6. The WRITE address is latched in at the end of cycle 2, and the data are written into the according memory location during cycle 5.

The signals $S1-S3$ denote select signals at register outputs of pipeline stages 1–3, as is indicated in Fig. 2. The signals $D3$, $D4$, and D_{out} are the multiplexed read data in the corresponding pipeline stages. In particular, $S3$ are signals appearing out of the input latches of the 1K block following the falling edge of PI (see below). $D3$ are the read data fed into the output latches of the 1K block that have to be stable at the falling edge of PO. Thus, approximately one clock period is available for the delay time of the 1K block.

The new scheme of selective clocking is also illustrated in Figs. 2 and 3. The registers within the third hierarchical level are clocked by nonoverlapping master/slave clocks PM and PS derived from a single external clock signal.
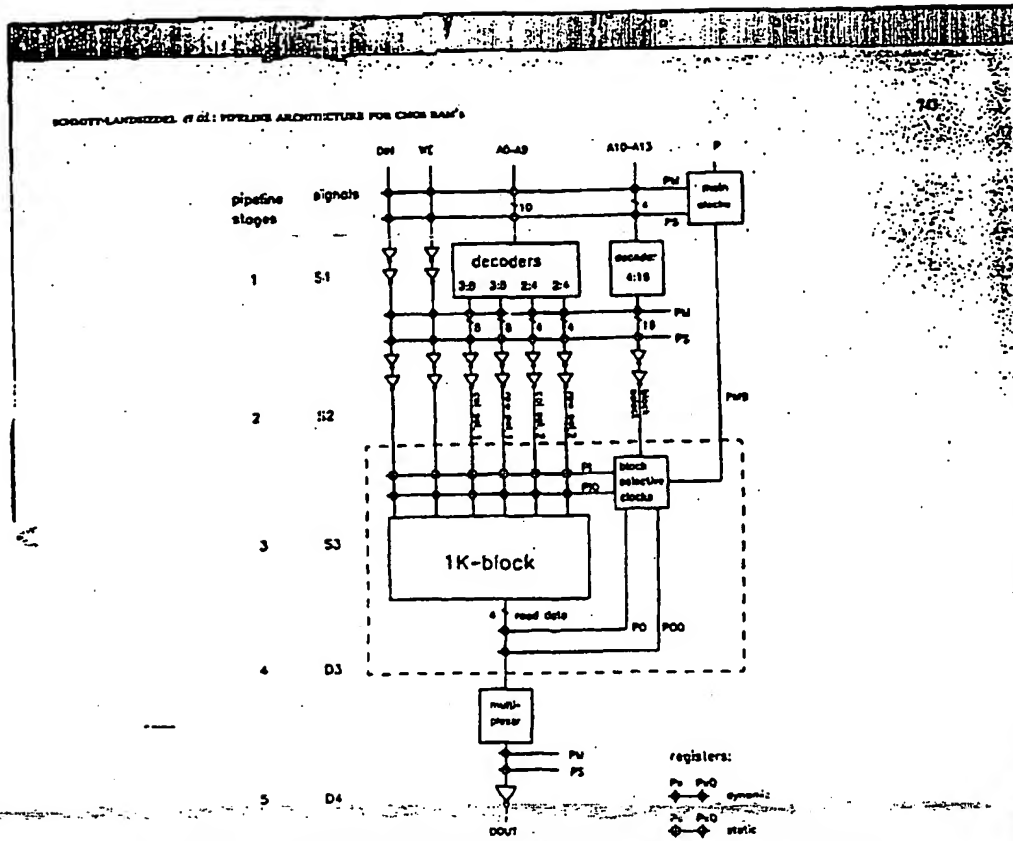
Fig. 2   Block diagram of a pipelined hierarchical 16K SRAM (PHSRAM). Circuits within dashed block are in each of the 16 subblocks.
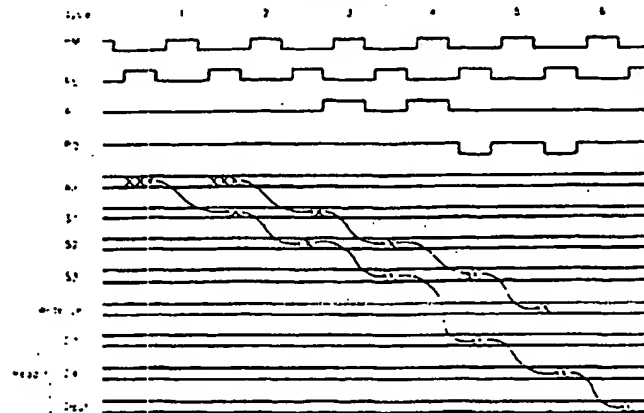

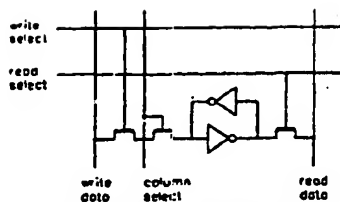Fig. 3.   Timing diagram of a read and write access to PHSRAM.

Fig. 4. Seven-transistor memory cell.

level 1   block of 64 memory cells

Fig. 5. Circuit diagram of 64-b block.

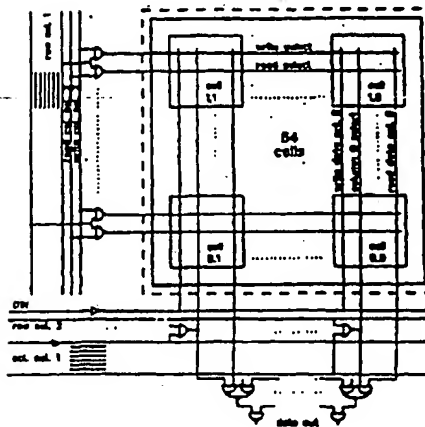Inverse clocks *PMQ* and *PSQ* are provided for the respective p-channel transistors of the transmission gates.

In the second hierarchical level, only one of the 16 1K blocks has to be activated at a time. Therefore, local clock signals are generated in each 1K block. The pair of input clocks *PI, PIQ* serves the input registers to the 1K block, whereas the output clocks *PO, POQ* are destined for the four data-out registers of the 1K block. To this purpose, a clock signal *PMB* having a slightly advanced phase with respect to *PM* is generated. It is gated with the block select signal, which is delayed by one clock period with respect to the output of the 4:16 decoder for the input clock generation. The select signal for the output clocks is delayed by another clock period.

Static registers are used for the input signals of the 1K blocks. When a block is not selected, the first latch of each input register is open, and the second latch is closed. So no undefined states can occur within an unselected subblock. On the other hand, in unselected output registers the second latch is open, and the inactive output lines are pulled down to low level. So the occurrence of concurrent signals is avoided in the fourth stage, where the data of several 1K blocks are multiplexed. With this scheme of selective clocking, the power consumption in the clock generators and subblock registers is drastically reduced as compared to global clocking.

IV. MEMORY CELL

A seven-transistor memory cell with separate word and data lines for READ and WRITE operation was developed (see Fig. 4). Due to the separate READ and WRITE data lines, no contrary data flow occurs in successive READ and WRITE cycles. Thus WRITE recovery times are avoided and it is possible to perform READ and WRITE operations at the same high clock frequency, and to switch between operating modes without wait cycles. A full logic voltage swing is obtained on the short data lines of the 64-b blocks, and a logic gate could be used for sensing instead of an area-consuming and slower sense amplifier. The memory cell size in a 0.8-μm technology is 14 μm×21 μm. For comparison, the size of a standard CMOS static cell in the same technology is 11 μm×17 μm.

Two WRITE access transistors are introduced in series connection. These access transistors are activated by a row select signal gated with the *WE* signal and a column select signal, respectively. Therefore, only a single memory cell is addressed in a 64-b block for writing instead of a whole row of cells. This eliminates noise margins due to charge sharing of neighboring cells in WRITE cycles. Consequently, the inverter sizes within the seven-transistor cell can be chosen to optimize READ access times. In particular, the size of the feedback inverter is minimum, similar to static latches, whereas the main inverter is designed to provide a strong cell signal. Precharging of the WRITE data lines is not necessary, and the data signal can be distributed to all cells in a block without further decoding. Thus the area overhead due to the additional access transistor and select lines is partly compensated by a reduction in decoding circuitry and omission of precharge circuits.

V. 64-b BLOCK

The circuitry of the 64-b block is shown in Fig. 5. All row and column select signals of level 1 and level 2 are active low. NOR gates form the row and column decoders. For a WRITE cycle, the WRITE-column select-2 signal and one of the row select-1 signals are active resulting in one active WRITE-row select (or WRITE word line). The row select-2 signal and one column select-1 signal activate one column select line. Data-in is distributed to all cells in the block. In only one cell are a WRITE-row select and a column select signal active at the same time.

During a READ cycle, the READ-column select-2 signal and one of the row select-1 signals activate one of the eight READ-select lines (or READ word line). The data bit
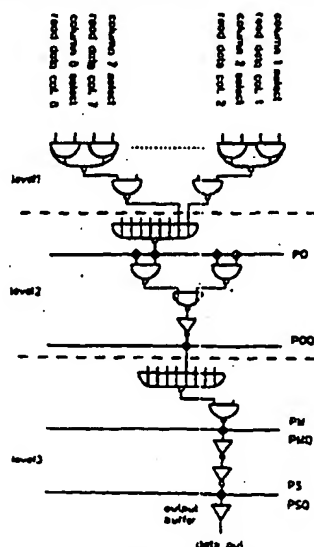
Fig. 6. Circuit diagram of readout circuits.

Fig. 7. Micrograph of experimental 4K block with hierarchical architecture.
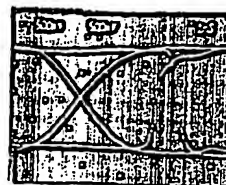
Fig. 8. Waveforms for READ access of 4K block. A delay of 1 ns is due to the on-chip pad driver.

from the addressed cell is passed through to one of the two data-out lines in a multiplexer logic, gated by the row select-2 and one of the column select-1 signals. Due to short READ data lines, there are no analog circuits necessary for sensing. A logic gate activated by the column select signal is sufficient to restore the readout signal to CMOS level.

As a main advantage, this structure offers fast access. It is also little sensitive to fluctuations in the fabrication process, as it contains only digital logic circuits. This also implies the possibility of transferring a design to a new technology by a simple shrink, without further redesign.

VI. READOUT CIRCUITS

The bus-driver tree proposed by Rem and Mead [4] can be easily recognized in the readout circuit of the 16K PHSRAM (Fig. 6). The number of lines that are multiplexed differs according to the level of hierarchy, the length of connected wires, and the kind of logic circuit used. The transistor widths increase with the level.

The eight output lines from the four 64-b blocks in a row of the 1K block are multiplexed in a pseudo-NMOS NOR gate. The output path is divided into two more pipeline stages, followed by an output buffer. The delays of the output stages 4 and 5 are shorter than in the critical stage 3, containing the 1K block. A certain delay has to be provided between latches in order to prevent

races. For an economic use of area, parts of the multiplexing circuits are placed between the register latches.

VII. RESULTS

Differently sized test structures for PHSRAM's were produced in a single-poly, double-metal 1-μm CMOS technology. Fig. 7 shows the micrograph of a 4K block consisting of 8×8 64-b blocks with seven-transistor cells. Its delay determines the clock rate of a 64K PHSRAM using 4K blocks as the critical stage. The measured waveforms for a READ access are shown in Fig. 8. The delay is 3.6 ns for the rising edge and 4.5 ns for the falling edge. This includes a delay of 1 ns caused by an on-chip pad driver, which does not contribute to the critical path delay. The falling edge delay of a corresponding 1K block plus driver is 3.5 ns. The delay of the measurement setup was compensated by measuring an on-chip short-circuit connection. The measurements are in good agreement with simulation results. The corresponding maximum clock frequencies were also estimated from simulations, accounting for the additional register delay. They are 200 MHz for 4K blocks and 250 MHz for 1K blocks. Note that the higher clock frequency possible with 1K blocks is paid for by larger area.

To judge the speed advantage of the PHSRAM, it is also interesting to compare the simulation results for

746

TABLE I
MEASUREMENT AND SIMULATION RESULTS FOR PHSRAM
TEST STRUCTURES

measured	1ns CMOS, 4k-block	3.3 ns
on-chip access time	1ns CMOS, 1k-block	2.3 ns
simulated	1ns CMOS, 4k-blocks	200 MHz
data rate	1ns CMOS, 1k-blocks	250 MHz
of PHSRAM	0.8ns CMOS, 1k-blocks	300 MHz

other architectures. For the 1-μm technology we estimate the access time of an asynchronous 64K SRAM with large cell arrays to be 14 ns. The introduction of the hierarchical architecture offers a 30% reduction of access time to 9 ns [5]. A corresponding synchronous SRAM with hierarchical architecture would have a minimum clock period of about 8 ns. The minimum period of 4 ns estimated for the pipelined RAM with 4K blocks is twice as fast.

Careful simulations were also performed based on the layout for a 16K×1 SRAM in a 0.8-μm technology with polysilicon gate, one pollicide, and two metal wiring layers. According to our computational results, the maximum clock frequency is above 300 MHz. The power consumption at this frequency is estimated to 0.8 W. The chip size is 16 mm^2. For comparison, the size of the corresponding asynchronous 16K SRAM with a standard six-transistor cell is 8 mm^2. The additional area of 8 mm^2 for the PHSRAM results in equal parts from using a fully digital hierarchical RAM architecture with seven-transistor cells and from the area required for registers and additional wiring for the pipelining. The measurement and simulation results are presented in Table 1.

VIII. SUMMARY

A novel architecture for fast CMOS RAM's was presented. The hierarchical architecture together with a seven-transistor memory cell provide a circuit using digital signal swings all over. Key advantages of the full-swing static logic circuitry are robustness with respect to fabrication tolerances and a high noise immunity. Moreover, the circuit can be reduced to finer structure sizes without any redesign, since there are no critical analog circuit parts. Pipelining the hierarchical architecture, a buffer RAM with a very high clock frequency and fully random READ/ WRITE capability can be realized. Trade-offs are larger area and the latency time of READ data with respect to the address input amounting to several clock periods.

Clock frequencies in the range of 300 MHz are possible for a 16K buffer SRAM in 0.8-μm CMOS technology. A 64K PHSRAM appears to be feasible with approximately the same clock rate. Comparable data throughputs can be achieved now only by BiCMOS and bipolar circuits in Si technology, but with a considerably higher power consumption or process complexity. The chip size of a 16K

BiCMOS SRAM [6] with similar performance is approximately 25% smaller. The general strategy of the pipelined hierarchical memory architecture is not restricted to CMOS static RAM's, but can also be applied to DRAM's or nonvolatile RAM's.

REFERENCES

[1] F. Miller, S. Remington, and R. Crisp, "High frequency system operation using synchronous SRAMs," in *Midcon/87 Conf. Rec.* (Chicago, ILL), Sept. 15-17, 1987, pp. 430-432.
[2] F. Towler *et al.*, "A 12th 6.5ns access/5ns cycle CMOS ECL static RAM," in *ISSCC Dig. Tech. Papers*, Feb. 1989, pp. 30-31.
[3] D. Schmitt-Landsiedel, B. Hoppe, G. Neuendorf, M. Wurm, and J. Winnerl, "Pipelined 16k buffer RAM with 300 MHz operating frequency," in *Proc. ESSCIRC* (Vienna), Sept. 1989.
[4] M. Rem and C. A. Mead, *IEEE J. Solid-State Circuits*, vol. SC-14, pp. 455-462, Apr. 1979; also in *Introduction to VLSI Systems*, C. Mead and L. Conway, Eds., 2nd ed. Reading, MA: Addison-Wesley, 1980, ch. 8.5.
[5] D. Schmitt-Landsiedel, G. Neuendorf, B. Hoppe, and H. J. Mausuch, "Hierarchical architecture for fast CMOS SRAMs," in *Proc. COMPEURO* (Hamburg, W. Germany), May 8-12, 1989.
[6] W. Heimsch, R. Kreta, and K. Ziemann, "A 4ns 16k BiCMOS SRAM," in *Proc. Symp. VLSI Circuits* (Kyoto, Japan), May 25-27, 1989.

Doris Schmitt-Landsiedel was born in Freiburg, West Germany, in 1952. She received the Dipl. Ing. degree in electrical engineering from the Technical University of Karlsruhe in 1977, the Dipl. Phys. degree from the University of Freiburg in 1980, and the Ph.D. degree in physics from the Technical University of Munich in 1984.

She was engaged in the areas of semiconductor lasers and nonlinear optics. In 1981 she joined the Corporate Research and Development Department of Siemens AG, Munich, West Germany, where she worked on scaling problems in MOS devices and design of high-speed MOS circuits. Currently she is in charge of a group involved in advanced DRAM circuit design.

Bernhard Hoppe received the master's degree and the Ph.D. degree in theoretical solid-state physics from the University of Frankfurt, West Germany, in 1982 and 1986, respectively. His dissertation was on phase transitions and critical phenomena in magnetic systems with highly degenerate ground states.

In 1986 he joined the Corporate Research and Development Department of Siemens AG in Munich, West Germany, where he was engaged in the design of high-performance CMOS logic circuits, neural networks, and mathematical methods for circuit optimization. Since 1989 he has been with VDO Adolf Schindling AG in Schwalbach, West Germany, where he is involved in the design of full-custom MOS circuits for automotive applications.

Gerd Neuendorf was born in Gross Twülpstedt, West Germany, in 1949. He received the Dipl. Ing.-degree in electrical engineering from the Technical University of Munich in 1985.

In 1985 he joined the Corporate Research and Development Department of Siemens AG, Munich, West Germany, where he was working in the design of high-speed MOS circuits and CMOS SRAM's and in the development of circuit optimization methods. In 1989 he joined the Semiconductor Division of Siemens AG, where he is presently involved in testing of complex CMOS circuits.

58

MIPS-X: A 20-MIPS Peak, 32-bit Microprocessor with On-Chip Cache

MARK HOROWITZ, MEMBER, IEEE, PAUL CHOW, MEMBER, IEEE, DON STARK, STUDENT MEMBER, IEEE,
RICHARD T. SIMONI, STUDENT MEMBER, IEEE, ARTURO SALZ, STEVEN PRZYBYLSKI, STUDENT MEMBER, IEEE,
JOHN HENNESSY, MEMBER, IEEE, GLENN GULAK, MEMBER, IEEE, ANANT AGARWAL, STUDENT MEMBER, IEEE,
AND JOHN M. ACKEN, MEMBER, IEEE

Abstract —MIPS-X is a 32-bit RISC microprocessor implemented in a conservative 2-μm, two-level-metal, n-well CMOS technology. High performance is achieved by using a nonoverlapping two-phase 20-MHz clock and executing one instruction every cycle. To reduce its memory bandwidth requirements, MIPS-X includes a 2-kbyte on-chip instruction cache. This cache satisfies 90 percent of all instruction fetches, and reduces the memory bandwidth of the processor by a factor of 2.5. MIPS-X has a peak operating rate of 20 MIPS, and provides an effective throughput of 12 MIPS when the effects of the on-chip cache, external caches, and pipeline stalls are included. MIPS-X contains 150K devices in an 8 × 8.5-mm² die.

To produce a high-speed computer system, MIPS-X uses a simple computer engine, a simple and fast clocking scheme, and a high-performance memory system. The simplicity of the basic processor allowed us to use a significant fraction of the design time and silicon area to integrate a part of the memory system on the processor. This paper provides an overview of MIPS-X, focusing on the techniques used to reduce the complexity of the processor and implement the on-chip instruction cache.

I. INTRODUCTION

THE MIPS-X project began in the Summer of 1984 with the goal of designing a second-generation RISC microprocessor that could be used as the processing node of a shared-memory multiprocessor. With the knowledge gained from early RISC designs [1]–[3] and the improved performance available from a 2-μm two-level-metal CMOS process we have designed a processor with a peak instruction rate of 20 MIPS. MIPS-X borrows from the original MIPS machine [1] the ideas of a simplified instruction set, pipelining, and a software code reorganizer to handle pipeline interlocks. However, to improve performance, MIPS-X uses a simpler instruction format, a deeper pipeline, an on-chip instruction cache, and a faster clock rate.

There are several areas that are important to consider when designing a high-speed processor, particularly one that is to be implemented in VLSI. These include the memory system design, the clocking methodology and the

complexity of the resulting hardware. We feel that the most important factor is simplicity. For a high-speed processor, additional functionality should only be added when it significantly improves the overall performance of the machine. The design team has a certain amount of time and silicon area it can use to complete its task. Resources spent implementing a feature are resources that cannot be spent on other aspects of the design. In MIPS-X, the execution portion of the processor occupies a small fraction of the die area, allowing us to use the extra area to improve the performance of another critical element of the processor, the memory system.

As instruction rates increase, the bandwidth and latency of the memory system become important issues. This is evidenced by the greater use of on-chip caches and instruction prefetch queues to decrease the average time required to access instructions [4]–[10]. Crossing chip boundaries has become a limiting factor in high-speed processor systems; this makes it difficult to access instructions and data quickly if they have to be kept off-chip. MIPS-X uses both a large 2-kbyte on-chip instruction cache and an external interface optimized for high-speed cache access to provide the required memory bandwidth for the processor.

Increased performance also implies faster clock rates and this makes the problem of clock distribution more difficult. Multiphase clocks exacerbate the situation because the time per phase is smaller and there are more phases to distribute. MIPS-X uses a simple two-phase clocking scheme, and locally generates additional clocks when necessary. Circuits using local clocks are often called self-timed because they derive the timing information from the delay of the circuit being controlled. The use of self-timed clocks makes the global clocking in MIPS-X simple but does add some circuit complexity in the parts of the chip that require additional clocks.

The next section gives an overview of the MIPS-X architecture and the supporting memory structure. This is followed by a description of the pipeline in Section III. Sections IV and V present the hardware required to implement this machine. Section VI follows with a description of the design methodology used to keep the hardware

relatively simple. The summary and .rent status of the project are given in Section VII.

II. ARCHITECTURAL OVERVIEW

The only instructions implemented in MIPS-X are those that contribute significantly to the performance of the machine. These instructions have been made to execute very quickly. The processor is a load–store machine: the only instructions that can access the word-addressed memory are explicit load and store instructions. All other instructions use the 32-word register file. There are three types of instructions: memory, branch, and compute. Memory instructions support a single addressing mode that adds an offset to the contents of a register to generate the effective address. Branch instructions contain an explicit comparison operation. This COMPARE AND BRANCH form was chosen to increase the speed of branches by removing the instructions that are normally needed to set the condition codes. Unlike some of the recently announced RISC machines [11], MIPS-X provides a full set of comparison operations for branches, rather than providing only simple (equality and sign) compares. Compute instructions are generally three-operand instructions with two sources and a destination. MIPS-X supports a wide variety of arithmetic, logical, and shift operations, including variable byte rotates to support character handling. A limited number of compute instructions include an immediate field, providing a simple way to generate and use ort 17-bit constants.

The instruction format was optimized for simple decode. All 37 instructions are 32 bits and use a fixed format for the register specifiers. The four formats can be seen in Fig. 1. The comp func field in the compute instructions directly feeds control inputs in the execute unit making decoding very simple or nonexistent.

MIPS-X requires a low-latency and high-bandwidth connection to memory. With single-cycle execution and a 20-MHz clock, the peak bandwidth required for instructions and data is 40 Mwords/s (160 Mbytes/s). Besides the difficulty in designing a memory system to support this data rate, transferring this amount of data across the pins of the package is extremely difficult. To reduce the large instruction bandwidth requirements, MIPS-X has a 2-kbyte instruction cache (ICache) on the processor. This cache occupies about one-half of the interior die area, satisfies roughly 90 percent of all instruction references [12], and reduces the instruction bandwidth requirements across the pins by a factor of six.[1] Missed instruction references and data references go off-chip to a large 64K-word external cache. The on-chip cache effectively dual ports the memory system, allowing the processor to simultaneously fetch data from the external cache and instructions from the internal cache. The large external cache is
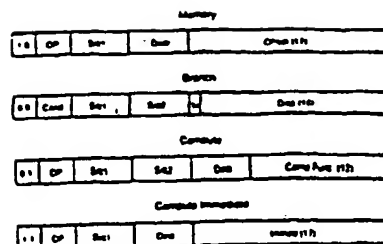
Fig. 1. MIPS-X instruction formats.

required to minimize the miss rate because accesses to the main memory take 15–20 processor cycles to fetch back four words. Low miss rates are also important to reduce bus contention in a shared-memory multiprocessor system.

The external interface is optimized for speed. It is designed to connect to a large cache memory, is fully synchronous, and can operate at a 50-ns cycle time. The interface is very simple: it presents an address by the beginning of a cycle and expects the data by the end of the same cycle. The general bus interface is placed on the other side of the external cache.

We realized early in the design that we would not be able to fit all the functionality needed for a high-speed computer onto a single die, so MIPS-X implements a simple, yet efficient coprocessor interface. This interface is made more difficult by the presence of the on-chip instruction cache which hides instructions from attached coprocessors. Instead of using valuable package pins to transfer the coprocessor instruction off the chip, MIPS-X uses the address and data bus for the coprocessor operations. During a coprocessor cycle an additional processor pin is asserted, indicating that the value on the address bus is a coprocessor instruction rather than a memory address. The coprocessors decode the instruction and determine their correct action. During these cycles the data bus can be used to transfer information between the coprocessor and MIPS-X. The inefficiency with this scheme is that all coprocessor-memory traffic must be transferred through the processor using extra instructions. We felt this would only be a significant problem for the floating-point processor. To improve the floating-point interface, two special memory instructions were added to MIPS-X that directly transfer data between one specific coprocessor and memory. With this minor addition we were able to provide a simple interface that supports high-performance coprocessors. One advantage of this interface is that coprocessor instructions look just like memory instructions and thus can be implemented easily.

MIPS-X provides separate system and user addresses. Programs running in user mode are prevented from accessing system addresses, while programs running in system mode can access either address space. The processor can enter system mode only by taking an interrupt or by

---

[1] During a miss two instructions are fetched during the two cycles when the processor is stalled. This leads to an average instruction fetch rate of one instruction every six cycles, or one-sixth of the original requirements.

executing a trap instruction. To support a dynamic paged virtual memory system, all instructions are restartable. The processor supports both maskable and nonmaskable interrupts. An interrupt causes the machine to flush the instructions in the execution pipeline, enter system mode, and jump to location zero. This simple support for exceptions provides the essential features needed to build an operating system for the processor.

III. PIPELINE

Instructions in MIPS-X require five clock cycles to complete: instruction fetch (IF), register fetch (RF), execute (ALU), memory access (MEM), and write back of registers (WB). During IF, the instruction is fetched from the on-chip instruction cache and loaded into the instruction register. The RF cycle is used to drive the register specifiers from the instruction register to the register decoders and then to perform the actual register fetch. During ϕ_1 of the execute cycle either the ALU or the shifter evaluates, and during ϕ_2 this result is driven onto the result bus. For branch instructions, the ALU is used to evaluate the branch condition, and a separate adder in the program counter unit is used to compute the branch destination. This adder has the same timing as the ALU and evaluates on ϕ_1 of ALU. For memory instructions, the ALU is used to compute the effective address and during ϕ_2 this address is driven to the address pads. By having the ALU evaluate in a single phase, the address has enough time to be driven off the chip before the end of the ALU cycle. Thus the address is valid at the pins of the chip when the memory cycle begins. This predrive of the address gives the external cache memory a full cycle (MEM) to complete its access. The result of the instruction is written into the register file during ϕ_1 of WB.

The MIPS-X processor is pipelined so that a new instruction can be started every cycle. Starting the next instruction before the current instruction is completed gives rise to a number of pipeline dependencies as shown in Fig. 2. For example, the result of a branch instruction is not known until the end of the ALU cycle, too late to affect the IF of the next two instructions. Therefore, the two instructions following a branch will be fetched independent of the outcome of the branch; the branch delay is two cycles. The pipeline also has a delay slot associated with loads. Since the data from the load does not enter the chip until the end of the MEM cycle, it arrives too late to be used in the ALU of the next instruction. The instruction following a load cannot use the value just loaded. The processor does not contain pipeline interlocks in hardware so these pipeline interlocks are handled by a pass of the assembler called the reorganizer, a technique pioneered by the original MIPS processor [1]. The reorganizer is responsible for generating a code sequence that is free from pipeline dependencies. If the reorganizer cannot find a useful instruction to put into a delay slot, it fills the slot with a no-op instruction, effectively stalling the machine for a cycle at the cost of increased instruction bandwidth.
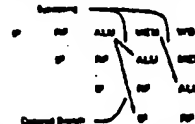

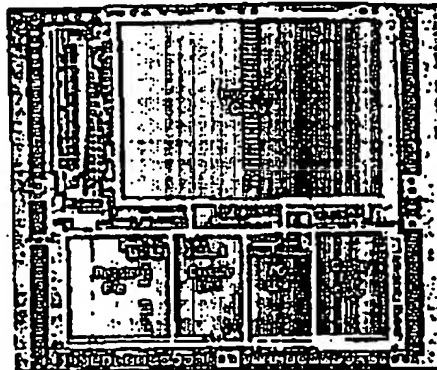Fig. 2. Pipeline dependencies in MIPS-X.

Fig. 3. Die photo of MIPS-X.

To help the software system use the two slots associated with a branch, MIPS-X can optionally squash (turn into no-ops) the instructions in the slots if the branch is not taken. This allows the reorganizer to predict that the branch will go and put the first two instructions of the branch destination after the branch. In this case the machine effectively starts executing the code at the branch destination right after the branch instruction. Only if the branch is not taken are these instructions turned into no-ops and the resulting cycles wasted.

To avoid having additional pipeline constraints, MIPS-X has two levels of internal forwarding or bypassing. The bypassing allows the result of one instruction to be used as input for the next instruction and is needed because the actual WRITE into the register file occurs late in the instruction, too late to be directly used in the next two instructions. The bypass logic slightly complicates the design of the register file, but greatly reduces the number of no-ops needed to eliminate interlocks.

IV. HARDWARE RESOURCES

A microphotograph of the processor with the major functional blocks outlined is shown in Fig. 3. The on-chip instruction cache dominates the die: occupying the upper half of the chip. The data path of the processor runs under the cache, and can be divided into four major sections. The register file contains 32 general-purpose 32-bit registers.
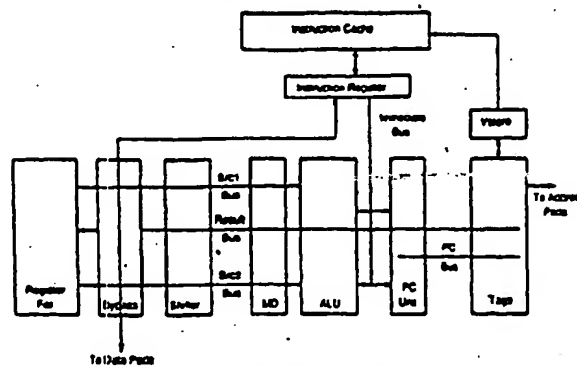
Fig. 4.   MIPS-X hardware resources.

the pipeline bypass registers, and the registers associated with the external memory interface. The execute unit contains a 32-bit funnel shifter, a 32-bit ALU, registers to support single-bit multiplication and division (MD), and the processor status word. In the program counter unit (PC-unit), there are two 32-bit adders, one used as an incrementer to calculate the next instruction address and the other used to compute the destinations of branches, and a chain of shift registers (PC chain) that is used to hold the addresses of the instructions currently in execution. These addresses are needed to restart the machine in the case of interrupt. The tag section contains the tags and valid bits for the on-chip instruction cache. Located between the data path and the ICache is the instruction register, which contains a set of pipeline staging registers, and a small amount of instruction decode logic. The instruction register is also responsible for writing instructions into the cache during an internal cache miss.

Fig. 4 shows the hardware and the major buses. Data are read from the register file on the Src1 bus and Src2 bus. Data are written to the register file from the bypass block. The result bus carries values to the bypass block and to the tag section where it is multiplexed with the PC bus and used as an address for memory instructions.

A. Instruction Cache

Much of the design effort of MIPS-X was spent implementing the on-chip instruction cache. The goal was to design a simple cache that provided a high hit rate and a low cache-miss penalty. The on-chip cache is organized as 32 blocks of 16 32-bit words. Each block has a tag indicating the part of memory that is currently stored in it, and each word in a block has a valid bit indicating whether this word is currently stored in the cache. The use of valid bits allows the cache to have a large block size but use sub-block replacement. The large block size was chosen to size the amount of storage required for tags, allowing

a full 512-word instruction cache to be placed on the die. The small number of tags also allowed the tag memory array to be placed in the data path, reducing the amount of wiring needed for the cache. With the tag array in the data path, the large ICache above the data path becomes a 512 × 32-bit static RAM.

The cache system has a full cycle for its access, but needs to determine whether the instruction will hit in the cache in a single phase. The early hit detect is needed to be able to use the next cycle to fetch the missed instruction from the external cache as shown in Fig. 5. The root of the problem is that external memory accesses really take one and a half cycles; the processor must drive the address pads on ϕ_2 of the cycle before the memory access. To fetch the missed instruction by the end of the first cache-miss cycle, the processor must drive the instruction address off-chip during ϕ_2 of the IF that misses, and thus we need the hit signal by the end of ϕ_1. Using the early hit detect, internal cache misses stall the machine for two cycles. The first cycle is used to fetch the missed instruction from the external cache, and the second cycle is used to write this value into the instruction cache. Since we assumed that the data from an external cache fetch are valid just before the end of the cycle, to reduce the miss delay to a single cycle we would need to extend the cycle time to provide sufficient time for a cache write to complete after the data become valid. Instead, MIPS-X uses the second cache-miss cycle to fetch from the external cache the next instruction that will be executed. Therefore, ICache misses have a penalty of two cycles, but fetch back two words. This fetch of two words halves the miss rate of the cache and provides roughly the same system performance as a cache with a single-cycle miss penalty, but accomplishes this performance without influencing the cycle time of the processor.

The tags are stored in a content-addressable memory using a standard ten-transistor CAM cell so they can be quickly compared against the current instruction address. Fig. 6 shows the tag array. The 32 tags are placed in the
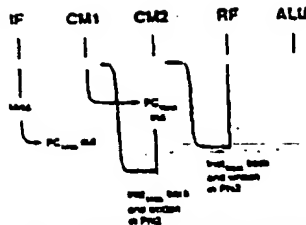
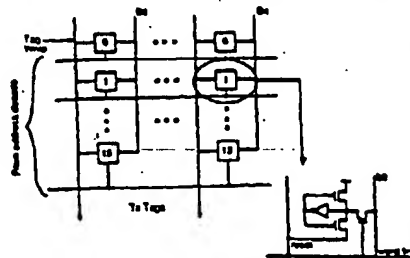Fig. 5. MIPS-X instruction cache-miss timing.

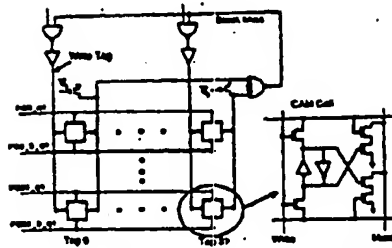Fig. 7. Valid store circuit.

Fig. 6. Tag array showing tag cell and self-timed write.

data path and match its pitch. Located above each tag are the 16 valid bits associated with that tag. The valid bit cells are the same width as the tag cells which means that the valid bit store (Vstore) fits directly on top of the tags.

Logically, the 32 tags are broken into four different sets of eight entries each. Low-order bits of the instruction address select a set and the associative compare is used to find the correct entry in the set. The most significant 24 bits of the instruction address are compared against the tag entries. The least significant 2 bits are the byte selector and are always zero for instructions; the next four bits select the correct word in each cache block, and the next two bits select the correct set.

Hit detection requires first comparing the current instruction address against the values stored in the CAM array, and then fetching the correct valid bit for the block that matches. To generate the hit information in one phase, the tag compare and valid bit fetch are performed simultaneously. The Vstore is logically organized as 64 words of eight bits. During the tag compare the low-order bits of the instruction address are used to index into the Vstore to fetch the eight possible valid bits, a bit for each tag that could match. Next these output lines are ANDed with the output of the tag comparison, and then ORed together to generate the cache hit signal. Since the tag compare and the Vstore access both require roughly 15 ns, it is easy to generate the hit signal in a single phase.

There are two types of internal cache misses: block miss and word miss, depending on whether the block for the

desired instruction is already in the cache. To make the cache-miss sequencer simpler, it only handles word misses. When a block miss occurs, a tag is written with the new instruction address and the valid bits for that tag are flushed in the same phase that the block miss was detected (Fig. 6). The tag write allocates a block for the instruction, and makes block misses look like normal word misses. To generate the write signal, we make use of the monotonic nature of the tag comparison logic. The match output of each CAM word is precharged on ϕ_2 and falls during ϕ_1 if the instruction address does not match. The outputs of all the match lines are NORed together forming the block-miss signal. This signal starts low at the beginning of ϕ_1 and rises only if a block miss occurs. It is used to drive the write line of the selected tag high, writing the current value of the program counter into the tag. Fig. 7 shows how the tag write line also serves as a virtual ground for the valid bits associated with that tag. When the write line is pulled high it forces all the cells to reset, clearing the valid bits for that tag.

MIPS-X uses a simple ring counter algorithm for selecting the tag to be replaced during a block miss. The ring counter is located above the Vstore, and is incremented after each block miss. The fetch of two instructions during a cache miss means that the ring counter must also increment when there is a block hit and word miss, and the ring counter points to the block where the hit occurred. This prevents a block miss during the fetch of the second instruction from clobbering a block that only had a word miss during the fetch of the first instruction.

The data portion of the instruction cache uses a fairly conventional static RAM design that has been optimized for synchronous operation. During ϕ_1 of IF, the bit lines and sense circuit of the RAM are precharged, and the low-order six bits of the instruction address are driven to the RAM and decoded. These six bits form the row address. Near the end of this phase, the tag comparison information is available and is sent to the RAM. This information is used for the column select. During ϕ_2 of IF, the selected word line is driven and the outputs of the sense amplifiers are latched into the instruction register. Because of the short bit lines and relatively large cell transistors, MIPS-X uses a simple unclocked sense circuit
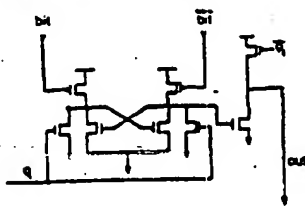
Fig. 8. RAM sense amplifier.

Fig. 9. CMOS dual-port register cell.

Fig. 10. Schematic of the self-timed bit-line WRITE circuit.

Fig. 11. Register bypass logic showing the comparators.

Fig. 12. Schematic of comparator circuit.

(see Fig. 8). This circuit is relatively slow since one bit line must fall below a p-FET threshold before it begins to sense, but has the advantages of not requiring a sense clock and not dissipating any static power. The measured access time of the RAM is about 18 ns, well within the single-phase access requirements.

B. Register File

The MIPS-X architecture requires a dual-READ single-WRITE register file, with support for double bypassing. The register file is time multiplexed, with WRITE's occurring on ϕ_1 and READ's on ϕ_2. To reduce the access time, three sets of decoders are placed above the register array, one for each access port. The inputs to the decoders are driven on the phase before their output is used so the decode time is not on the critical path for accessing the registers.

The initial design of the register cell used dual-differential buses, but this was dropped because the short bit lines le sense amplifiers unnecessary. Instead we used a CMOS version of the six-transistor RAM cell with split word lines described by Sherburne et al. [2]. Fig. 9 shows the CMOS cell. Time multiplexing the register array did pose a minor problem since both bit lines must start at 5 V for a READ. The self-timed circuit shown in Fig. 10 was used to solve this problem. This circuit detects when a WRITE has completed, turns off the WRITE and then restores both bit lines high. A row of dummy cells was placed above the register array; these cells are hardwired to always contain a zero. Thus, after a READ the dummy bit line is always low and the $\overline{bit}$ line is high. The write drive for the dummy row input is tied high, so it always tries to write a ONE into the cells. Transistor M_{sense} detects when the bit lines have crossed by enough to write the register. This transistor discharges the precharged node $\overline{Done}$, causing $Done$ to rise, and forcing the write drivers to recover the bit lines for the following READ. Transistor M_{rec} is needed to prevent the circuit from oscillating. If it is deleted, then the recovery of the $\overline{bit}$ line will cause $\overline{Done}$ to rise, and the write will restart. The write and recovery is quite fast, requiring less than 20 ns to complete.

To remove many potential pipeline interlocks, the register file is double bypassed. This requires adding bus drivers to two latches in the data path, and adding four comparators in the control as shown in Fig. 11. The comparators check the destination of the previous two
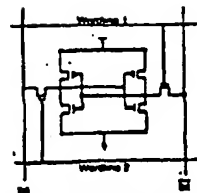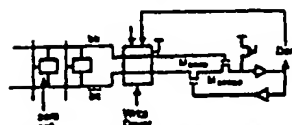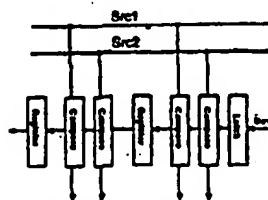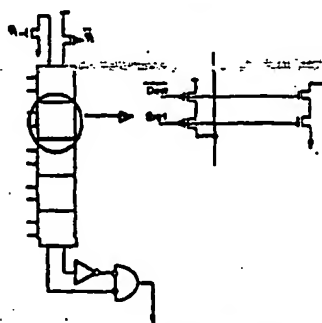
instructions against the two register sources of the current instruction to see if bypassing is required. If a match occurs, the correct latch output is driven onto the source bus instead of the data from the register file. The comparators are built around the set of latches needed to delay the destination specifier, which is driven into the register file on ϕ_1 of RF but not used until ϕ_1 of WB. The comparators use a precharged gate of n transistors and a predis-

charged gate of p transistors to avoid requiring both the true and complement versions of the register specifiers. Fig. 12 shows the comparator circuit.

V. CONTROL

The simple instruction format and the use of a lock-step pipeline make the control for the processor relatively simple. Most of this control is implemented in simple decoders and PLA's located above the part of the data path being controlled. To keep the complexity of this logic low, each designer was responsible for the design of a section of the data path and its control. This organization provided incentive to arrange the overall design to minimize the amount of random logic needed.

A. Global Control

Care was taken to keep the global control for the machine extremely simple. There are only three types of pipeline interruptions possible—exceptions, external cache misses, and internal cache misses—and of these only the first one requires the pipeline to be flushed. The cache misses only cause the processor to stall until the required data become available. In the case of an exception (either an interrupt, external fault, or internal fault) MIPS-X holds the instruction addresses of the last four instructions in the PC chain, squashes the instructions in the pipeline by preventing them from writing their results back into the register file, and jumps to system address 0. No attempt is made to complete instructions in the pipeline that occur before the instruction that caused the exception. The uniform effect of an exception makes the controller quite simple. The exception signal directly no-ops the instructions in the MEM and ALU phase of the pipe, and also sets up a small finite state machine (FSM) that causes the instructions in IF and RF to be converted to no-ops. The machine can be restarted by simply jumping to the addresses of the instructions stored in the PC chain.

The FSM used for exceptions is also used to implement the conditional evaluation of the two instructions that follow a branch. If the branch does not go, then during its ALU cycle the input to this FSM is set converting the instructions in RF and IF (the two slots of the branch) into no-ops. The squashing of branch slots does not need any additional logic; the same hardware is used to implement exceptions.

An FSM for handling internal cache misses is the only other global control that MIPS-X requires. During an ICache miss this controller sequences the machine through the two cache-miss states before resuming the execution pipe. The two FSM's are shown in Figs. 13 and 14. Collectively, these two controllers use less than 0.2 percent of the total chip area and are built with standard cells.

B. Stalling the Processor

The pipeline is stalled by using a set of qualified ϕ_1 clocks, ψ_1 and ψ_{1PC}. These clocks are used to latch all the
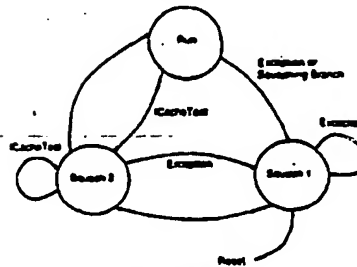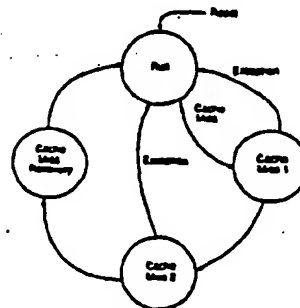

Fig. 13.   Squash FSM.

Fig. 14.   Cache-miss FSM.

state information in MIPS-X, and the machine is stalled by simply preventing these clocks from rising. This scheme is similar to one used to stall a floating-point unit in the case of an unusually long carry [13]. If ψ_1 does not rise, the machine throws away the results computed on ϕ_1, and during the next ϕ_2 repeats the ϕ_2 operation of the previous cycle.

The ψ_1 clock is ϕ_1 qualified by $\overline{external cache miss}$ and internal cache miss. The ψ_{1PC} clock is ϕ_1 qualified by only external cache miss. Two clocks are needed since part of the processor must be clocked during internal cache misses. In particular the cache-miss FSM. This set of logic uses ψ_{1PC} while the rest of the chip uses ψ_1.

The ψ_1 clocks can only be used as an input to a latch; the clocking of functional units is always done on the true clocks ϕ_1 and ϕ_2. This allows the ψ_1 clocks to be slightly shorter than ϕ_1, or said a different way, it means that the external cache-miss signal can arrive a little late. As long as the external cache-miss signal monotonically falls, it can actually arrive at the processor after the end of the MEM cycle, during ϕ_1 of WB. The external miss signal can arrive up to 10 ns late and still provide a valid ψ_1 clock. This gives the external cache about 10 ns to generate the miss signal after the data fetch, and prevents the cache tag comparison from being on the critical path for memory accesses.

VI. DESIGN METHODOLOGY AND TESTING

The basic MIPS-X architecture and pipeline structure were developed during the first six to nine months of the project. During this time an instruction level simulator for the machine was developed and used to evaluate the effects of different architectural features. We also investigated many organizations for the internal cache memory before settling on the one described in this paper. The different architectural trade-offs are described in more detail in [14]. In parallel with the architectural definition, we began investigating different implementations, and about a year after the project started we had a paper design of the hardware needed to implement the processor. The paper design included the layout of a number of the large structures MIPS-X would need. The layouts were done to get a better feel of the density and performance of the CMOS technology that we would use. At this point the chip was partitioned into several major functional units: the register file, the execute unit, the PC unit, the tag store, the instruction cache, the instruction register, and the external interface. Each of these sections, including its control, was designed by a single person. There was a total of six people. We used a tall thin design style: each designer was responsible for the design of a section, from writing the functional description for simulation, to generating the layout. Interface signals between the sections were fixed at the start of this design phase and then negotiated between the various parties if changes were required.

The first step of the detailed design was to write a functional description for the machine. We chose to write a custom simulator in Modula-2 because we lacked a good functional simulator at the time. The individual sections were written first, debugged, and then put together when everyone was satisfied that their sections were working correctly. This functional simulator became the *de facto* definition of the machine and was used quite extensively in the verification of the layout and testing of the silicon.

Once the functional definition was complete, the layout effort was started in earnest, using the Magic [15] layout system. This system has incremental design-rule checking and hierarchical extraction. Each section was extracted and then simulated using RSIM [16], a switch-level simulator. The functional simulator was modified so that it could be used to drive the RSIM simulations, making the verification of the circuits much easier. The functional simulator would provide the input vectors to a switch-level model of a subsection of the chip, and check the outputs of the switch-level simulator. This proved to be a powerful tool because it made it very easy to find differences between the functional and circuit representations. Using this method each designer was able to verify his section against the functional simulator before releasing it to the full chip simulation. On a MicroVax II, simulation of the entire ... ulation (without the cache array) took about one minute per clock cycle and only found a few errors. Most were subtle timing errors that the functional simulator could not catch.

Five machines were kept busy for about two weeks to do the final simulations before tape-out.

To simplify the testing of MIPS-X, we included the ability to separately test the processor and the large instruction RAM. By asserting an external pin the cache can be disabled, allowing the processor to run even if the cache is not functional. This feature simply forces a cache miss on every cycle so that the cache is never accessed. Asserting another pin puts the processor in the cache test mode. In this mode, the PC unit generates sequential addresses while the data bus is connected to the cache so that the cache can be directly read and written. These testing pins were used quite extensively during testing.

No special hardware was needed to test the data path of the processor. Whenever the processor is not handling an internal cache miss, the address pins are driven to be the value of the result bus. This makes it easy to observe the result of compute instructions and check the functionality of the execute unit and the register file.

Some hardware was added to make testing of the data-path control easier. By placing a small amount of logic under the data bus we could directly observe groups of control pins on the data pads by asserting a test pin. This can be done in the middle of any clock phase to allow direct observation of the internal control state of the machine. So far, this feature has not been used because no problems have been found in the control.

VII. SUMMARY AND STATUS

The first version of the MIPS-X processor was sent out for fabrication in May of 1986, and silicon was returned at the beginning of October. The functional simulator was used to generate test vectors for a low-speed functional tester developed at Stanford University. Simple speed testing was done by loading a small program into the instruction cache with the tester and then turning up the clock speed while observing the address bus. These parts are fully functional, run at 16 MHz, and dissipate less than 1 W at nominal operating conditions. Although the parts did not meet our ultimate cycle-time specification, they did run as fast as the simulation predicted. These die have been probed using low-capacitance probes and the waveforms match the simulation results quite well. The slower speed is caused by a slow path involving branches that has been fixed on the next revision of the part. This revision also includes a number of other small changes to improve other slow paths, and to make the external interface easier to use. We fully expect this version of the part to meet the 50-ns cycle time. We are also working on a simple shrink of the part to a 1.6-μm CMOS technology. This will yield a die of under 6.5 mm on a side, with a cycle time of over 25 MHz.

MIPS-X demonstrates the power of keeping VLSI processors simple, obtaining an effective throughput of over 10 MIPS while using a conservative technology and a relatively small die size. The key was to use the silicon

778

where it made the most difference: in the memory system design.

REFERENCES

[1] C. Rowen, S. A. Przybylski, N. P. Jouppi, T. R. Gross, J. D. Shott, and J. L. Hennessy, "A pipelined 32b NMOS microprocessor," in ISSCC Dig. Tech. Papers, Feb. 1984, pp. 180-181.

[2] R. W. Sherburne Jr., M. G. H. Katevenis, D. A. Patterson, and C. H. Sequin, "A 32b NMOS microprocessor with a large register file," IEEE J. Solid-State Circuits, vol. SC-19, pp. 682-689, Oct. 1984.

[3] G. Radin, "The 801 minicomputer," in Proc. SIGARCH/SIGPLAN Symp. Architectural Support for Programming Languages and Operating Systems. ACM (Palo Alto, CA), Mar. 1982, pp. 39-47.

[4] M. Hill et al., "Design decisions in SPUR," Computer, vol. 19, no. 10, pp. 8-22, Oct. 1986.

[5] D. MacGregor, D. Mothersole, and B. Moyer, "The Motorola MC68020," IEEE Micro, vol. 4, no. 4, pp. 101-118, Aug. 1984.

[6] K. A. El-Ayat and R. K. Agarwal, "The Intel 80386—Architecture and implementation," IEEE Micro, vol. 5, no. 6, pp. 4-22, Dec. 1985.

[7] D. Phillips, "The Z80000 microprocessor," IEEE Micro, vol. 5, no. 6, pp. 23-36, Dec. 1985.

[8] A. Berenbaum, B. W. Colbry, D. R. Ditzel, R. D. Freeman, H. R. McLellan, M. Shoji, and K. J. O'Connor, "A pipelined 32b microprocessor with 13Kb of cache memory," in ISSCC Dig. Tech. Papers, Feb. 1987, pp. 34-35, 131.

[9] P. W. Bosshart et al., "A 553K-transistor LISP processor chip," in ISSCC Dig. Tech. Papers, Feb. 1987, pp. 202-203, 402.

[10] D. Archer et al., "A 32b CMOS microprocessor with on-chip instruction and data caching and memory management," in ISSCC Dig. Tech. Papers, Feb. 1987, pp. 32-33, 320.

[11] J. Moussouris et al., "A CMOS RISC processor with integrated system functions," in Compcon Dig. Papers, Mar. 1986, pp. 126-131.

[12] A. Agarwal, P. Chow, M. Horowitz, J. Acken, A. Salz, and J. Hennessy, "On-chip instruction caches for high-performance processors," in Proc. 1987 Stanford Conf. Very Large Scale Integration (Stanford, CA), Mar. 1987, pp. 1-24.

[13] G. Wolrich, E. McLellan, L. Harada, J. Montanaro, and R. Yodlowski, "A high performance floating point coprocessor," IEEE J. Solid-State Circuits, vol. SC-19, pp. 690-696, Oct. 1984.

[14] P. Chow and M. Horowitz, "Architectural tradeoffs in the design of MIPS-X," in 14th Annual Int. Symp. Computer Architecture Conf. Proc. (Pittsburg, PA) June 1987, pp. 300-308.

[15] J. K. Ousterhout, G. T. Hamachi, R. N. Mayo, W. S. Scott, and G. S. Taylor, "The Magic VLSI layout system," IEEE Design Test Comput. vol. 2, no. 1, pp. 19-30, Feb. 1985.

[16] C. Terman, "Simulation tools for digital LSI design," Massachusetts Inst. Technol., Cambridge, Tech. Rep. MIT/LCS/TR-304, Sept. 1983.

Mark Horowitz (S'77-M'78) received the B.S. and M.S. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, in 1978, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1983.

Since September of 1984, he has been an Assistant Professor in the Department of Electrical Engineering at Stanford University. His current research interests are in the area of high-performance digital system design, especially integrated circuit design, and computer tools to aid in the design process. His research projects have ranged from high-speed ECL RAM design to designing a 20-MIPS CMOS processor to improved models for switch-level simulation.

In 1984 Dr. Horowitz was given a Presidential Young Investigator Award, and an IBM New Faculty Development Award.

Paul Chow (S'79-M'83) is from Peterborough, Ont., Canada. He received the B.A.Sc. degree with honors in engineering science, and the M.A.Sc. and Ph.D. degrees in electrical engineering from the University of Toronto, Toronto, Ont., in 1977, 1979, and 1984, respectively.

In 1984 he joined the Computer Systems Laboratory at Stanford University, Stanford, CA, as a Postdoctoral Fellow and became a Research Associate in April of 1987. He has been working on the MIPS-X project. His current research interests include computer architecture, the design of large VLSI systems and the CAD tools involved.

Don Stark received the B.S. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge, in 1985 and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, in 1987. Currently a graduate student at Stanford, his interests include computer architecture, VLSI, and CAD tools.

Mr. Stark is a member of Tau Beta Pi and Eta Kappa Nu.

Richard T. Simoni received the B.S. degree in electrical engineering from Rice University, Houston, TX, in 1984 and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, in 1985. He is now a graduate student in the Department of Electrical Engineering at Stanford University, where his research interests include multiprocessor memory-hierarchies, process scheduling algorithms for multiprocessors, and system-level computer aided design.

Mr. Simoni is currently a National Science Foundation Fellow.

Arturo Salz received the B.S. degree with honors in electronic and communication engineering from Universidad Iberoamericana, Mexico, 1982 and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, 1985.

He is currently a Ph.D. student in the Department of Electrical Engineering at Stanford University. His research interests include high-performance processors, multiprocessor architectures, computer-aided design, and interfaces.

From 1981 to 1983 he was Director of Technology Applications, Microdat S.A.

Steven Przybylski was born in February 1959 in Ottawa, Ont., Canada. In 1980 he received the B.A.Sc. degree with honors upon completing the Engineering Science programme at the University of Toronto. He received the M.S.E.E. degree from Stanford University, Stanford, CA, in 1982 and is currently working towards the Ph.D. degree, also at Stanford University. Between 1983 and 1984 he was a major contributor to the MIPS project at Stanford.

During 1985 he was with MIPS Computer Systems, of Sunnyvale CA, where he participated in the architectural definition and VLSI implementation of the R2000 CPU.

John Hennessy (S'73–M'77) received the B.E. degree in electrical engineering from Villanova University, Villanova, PA, in 1973. He received the Masters and Ph.D. degrees in computer science from the State University of New York at Stony Brook in 1975 and 1977, respectively.

Since September 1977, he has been in the Computer Systems Laboratory at Stanford University, Stanford, CA, where he is currently a Professor of Electrical Engineering and Computer Science, and Director of the Computer Systems Laboratory. He initiated the MIPS project at Stanford in 1981. During a leave from the academic life in 1984–1985, he cofounded MIPS Computer Systems and acted as Chief Scientist. His current research is in the area of high-performance architectures and software for such machines.

Dr. Hennessy is the recipient of the 1983 John J. Gallen Memorial Award and a 1983 Presidential Young Investigator Award.

Glenn Gulak received the B.A.Sc. degree in electrical engineering from the University of Windsor, Windsor, Ont. Canada, in 1978. While on an NSERC postgraduate scholarship, he received the M.Sc. and Ph.D. degrees in electrical engineering from the University of Manitoba, Winnipeg, in 1980 and 1984, respectively.

From February 1985 to January 1986 he worked on the MIPS-X project while on an NSERC University Research Fellowship. Since April 1987 he has been a Research Associate in the Information Systems Laboratory at Stanford University, Stanford, CA. His research interests are in digital communications, signal processing algorithms, computer architecture, and VLSI.

Anant Agarwal received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Madras, India, in 1982, and the Masters and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1984 and 1987, respectively. He is currently a post-doctoral research affiliate with the Computer Systems Laboratory at Stanford, where he worked on the architecture and design of the MIPS-X processor. His research interests include high-performance computer architectures, performance analysis and modeling, VLSI, and CAD tools.

John M. Acken (S'73–M'77) received the B.S. and M.S. degrees in electrical engineering from Oklahoma State University, Stillwater, and is currently working on the Ph.D. degree in electrical engineering at Leland Stanford Junior University, Stanford, CA.

From 1970 through 1973 he served in the armed forces as an electronics repairman. From 1978 to 1983 he was a member of the Computer-Aided Design Division of Sandia National Laboratories. Since 1983 he has been at Stanford University working on fault modeling for CMOS custom logic integrated circuits.

SESSION VI: MOS TECHNIQUES

WPM 6.5: Chip and System Characteristics of a 2048-Bit MNOS-BORAM LSI Circuit

Robert J. Lodi, H. A. Richard Wegener, Bernard B. Koziki, Millicent Borovick, William L. Moberg and Roger Newman

Sperry Research Center

Sudbury, MA

Charles A. Beitz

Sperry Univac

St. Paul, MN

AN OPERATING MODULE of a new memory system — a Block-Oriented Random-Access Memory (BORAM)[1] utilizing nonvolatile semiconductor MNOS[2] information storage — has been developed*. A fully populated 295K-bit module, it is organized in 32 blocks of 256 words, each word being 36 bits wide. The major performance parameters of this system are its data transfer rate of one word per 150 ns block access time of 2 µs, block read cycle time of 42 µs, and write cycle time of 2 ms. The heart of this module is a 2048-bit MNOS-LSI memory chip designed from the beginning to meet the system requirements.

The MNOS-BORAM chip has been fabricated by a simple extension of a typical P-channel MOS process. Isolation between MNOS memory transistor array and peripheral circuitry was achieved by a P-diffused wall through an N-type epitaxial layer on a P-type substrate. Another diffusion is added to provide N⁺ contacts to the isolated N region. The memory transistor has a stepped-gate structure, resulting in fixed threshold devices that have both a thick oxide layer and a nitride layer for their gate dielectric. Since each gate type requires a separate masking step, there are eight masking steps necessary to manufacture this chip. Only the normal single layer metallization of aluminum is used. The layout rules are relatively generous, with an average of 0.4 mil of minimum widths and spacings in use. In the design approach, essentially static (resistance ratio) logic is the rule. This requires meticulous care to meet performance requirements in spite of the size of the chip (198 x 186 mils²).

The memory chip (Figure 1) is essentially an EAROM organized in 32 randomly addressable blocks, each having 64 serially accessed bits which represent the same bit of 64 different words. The circuit elements providing the access are two 32-bit two-phase static shift registers, one communicating in parallel with all the bits of the odd words, the other with all the bits of the even words. Each shift register

*Project completed under NADC Contract No. N62269-73-C-0953, with R. Fedorak as project engineer, and models delivered to Naval Air Development Center.

[1] Beitz, C. A., "MNOS-BORAM Development" Government Microcircuit Applications Conference Digest, p. 20-21; June, 1974.

[2] Sewell, F. A., Wegener, H. A. R., Lewis, E. T., "The Variable Threshold FET: Theory and Experiment", ISSCC Digest of Technical Papers, p. 182-183; Feb., 1969.

performs well up to a clock frequency of 1 MHz. A multiplexer interleaves the odd-even data stream at the one I/O pin, resulting in a maximum data rate of at least 2 Mb/s. An off-chip four-into-one multiplexer ultimately converts the 64-bit sequence from four parallel chips into the 256-word block transferred at four times the frequency.

The writing operation requires two steps. During the first an addressed block represented by 64 memory transistors on each chip is erased by the application of a 1 ms 30-V pulse. The second step utilizes the information previously placed in the shift registers to inhibit the threshold voltage change in predetermined locations of the addressed block, when a 1-ms, 30-V write pulse is applied.

Testing has indicated that retention of information is orders of magnitude of years, when a specific chip is in a nonaddressed condition at 25°C; Figure 2. When a worst-case stuck-in-one address situation exists, retention is in excess of one year; Figure 3.

While for present purposes, the chip is mounted in a 40-lead ceramic dual in-line package; the number of functional contacts is 25. Their purposes and their supply voltages are sketched in Table I. Table II describes nominal timing requirements.

About a thousand fully functional chips have been evaluated and their behavior within a system has been documented. Thus both knowledge and confidence have been accumulated for this combination of MNOS-LSI process, MNOS transistor family characteristics, and circuit design approach. As a result it is currently being used in the development of two separate MNOS-LSI chips with the characteristics of static MOS P-channel RAMs, but with nonvolatile information storage.
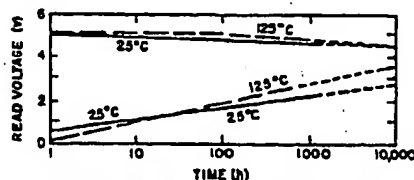

FIGURE 2—Plot of read voltage at which first bit of addressed block failed versus storage time of unaddressed chip. Top lines represent one logic state of the MNOS memory transistors at indicated temperatures; bottom lines represent their logic complement. End of retention is defined by a read voltage difference between the two levels of less than 1V.

Name	Function of pin
MI	Memory inhibited from read and write when MI high (zero V)
MP	Memory polarity control
	MNOS gate to substrate positive for MP = +12 V
	MNOS gate to substrate negative for MP = +0 V
R/W	Read-write select
	Read for R/W low
	Write for R/W high
Z9	Contact for external diode circuit such that (Z9 = VW + S2)
VCC	+5 V dc for data output
DO	Data output line
O/E	Odd or even register output select; Odd for O/E low, Even for O/E high
GND	Zero Volts dc
VGG	−18V during read and write
OLC	Odd register latching clock
VSS	+12 Volts dc
DI	Data Input (serial)
S1	Substrate (N-epi) contact for fixed-gate circuitry = VSS +3
OTC	Odd register transfer clock
RL	Read load control (memory to registers)
S2	Memory substrate (N-Tub) and P-isolation contact = VW
VW	Write voltage = VGG−VSS
ETC	Even register transfer clock
ELC	Even register latching clock
VR	Read voltage = VSS−9V
A0 TO A4	Block address inputs

TABLE I—The pin designations of MNOS-BORAM chip.

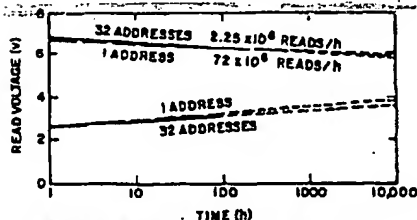| Specified Rise and Fall Times (10% to 90%)			
	t_{rise}	t_{fall}	
S2, VW	100 μs	10 μs	±20%
VGG, S2, VR	100 ns	100 ns	±25%
ALL OTHER INPUTS	60 ns	60 ns	±50%

TABLE II — Nominal timing

FIGURE 3—Plot of read voltage at which first bit of addressed block failed versus time of continuous serial reading of all blocks on one chip (32 addresses), and versus time of continuous reading of the same block on one chip (1 address). See Figure 2 caption for more details.

1976 IEEE INTERNATIONAL

SOLID-STATE CIRCUITS CONFERENCE

DIGEST of TECHNICAL PAPERS

First Edition

February, 1976

1976 IEEE International Solid-State Circuits Conference

DIGEST OF TECHNICAL PAPERS

TK 7870
I 56
1976

PRINTED IN THE UNITED STATES OF AMERICA

Editor: Lewis Winner

Technical Editor: J. A. A. Raper   Associate Editor: B. K. Winner

CIP   MAR 4 1976

76

SESSION III: NONVOLATILE AND APPLICATION-SPECIFIC MEMORIES

WAM 3.5:   A Dual-Port 85ns 64Kx4 DRAM with a 50MHz Serial Output

Frank Whiteside, Tom Manion, Roy Sirby

Mostek Corp.

Carrollton, TX

A DUAL-PORT 64K x 4 DRAM with a 50MHz 256 x 4 serial output has been designed for graphics applications. The device has a random access time of 65ns, a serial access time of 15ns, and includes an individual bit write mask, a continuous serial data stream, a selectable serial access start location, and an internal refresh counter[1]. Additional features include flash write, internal address counters, a gated serial clock, and column redundancy. A block diagram of the architecture is shown in Figure 2.

In the flash write mode, 256 x 4b are written in a single cycle to a 4b pattern defined by a write register. The entire matrix can thus be written in 256 cycles (~65μs). Because rapid flash write cycles could cause a large voltage swing on the $V_{CC}/2$ memory cell plate, an active driver circuit is used to maintain the cell plate voltage.

The cell plate driver circuit uses a $V_{CC}/2$ voltage divider and a unity-gain operational amplifier with a class AB output stage. Standby current is less than 100μA. Because a large capacitive load is being driven, extra care is required to avoid loop stability problems. A form of lead-lag compensation is used to limit phase shift near the unity loop gain frequency; Figure 3. Open loop voltage gain is approximately 50dB, as gain was traded off for stability and fast transient response.

Available too are a loadable row address counter and a column address register. These can be used to cycle through the memory sequentially (without using external addresses), while loading the serial output register or performing flash write.

To allow faster operation, the serial clock is gated internally by an enable pin. When disabled, the device can be clocked continuously without losing the serial bit position or consuming active power. The synchronous enabling scheme allows the use of fully dynamic output circuits, providing faster access times for a given power consumption.

Four rows and sixteen columns of laser implemented redundancy are provided. Column redundancy is difficult on a device with a serial output port, because the spare column must map onto the correct location in the serial data stream, as well as onto the correct logical address. There are two basic methods of realizing column redundancy for the serial port: the data from the spare column may be steered to the serial register locations while the register is being loaded, or the data from the spare columns may be stored in a separate register, and swapped into the serial data stream in real time. Because speed was deemed more important than die area for this device, the first method was chosen. The redundant columns are connected to the proper serial register locations via a data bus and laser programmable connections; Figure 3.

Because of the redundancy procedure used, the serial register and output bus are single-ended, rather than complementary to limit the number of wires running across the die. A capacitively-matched dummy output line and dummy register cell are used as a reference when sensing the serial output bus.

In a DRAM with fast asynchronous outputs, particular attention must be paid to noise immunity. All outputs have controlled di/dt characteristics to achieve fast switching times with tolerable noise; Figure 4. A controlled risetime is used on the active restore signal, and the precharge current is reduced by the divided bitline $V_{CC}/2$ architecture[2]. Two levels of metal are used for good bussing integrity.

The divided bitline matrix architecture combined with V_{CC} sensing substantially reduces the CV^2f power of the matrix and results in a 50mA DRAM operating current for a 175ns cycle time. Word lines and column select lines are boosted above V_{CC} to allow a full V_{CC} level to be written or restored into the cell. Active restore occurs immediately after sensing.

The device has been fabricated using a 1.5μm double-level metal NMOS process. The memory cell capacitor oxide thickness is 150Å. The cell plate voltage is held to $V_{CC}/2$ to reduce the oxide stress and to improve V_{CC}-bump performance[3]. The memory cell size is 4.5μm x 12μm. Transistors have 275Å gate oxides and are fabricated using a sidewall spacer oxide technique to form an LDD structure.

[1] Ishimoto, S., et al., "A 256K Dual Port Memory", ISSCC DIGEST OF TECHNICAL PAPERS, p. 38-39; Feb. 1985.

[2] Taylor, R. and Johnson, M., "A 1Mb CMOS DRAM with a Divided Bitline Matrix Architecture", ISSCC DIGEST OF TECHNICAL PAPERS, p. 242-243; Feb. 1985.

[3] Edmoston, R., et al., "A 100ns 256K DRAM with Page-Nibble Mode", ISSCC DIGEST OF TECHNICAL PAPERS, p. 228-229; Feb. 1983.

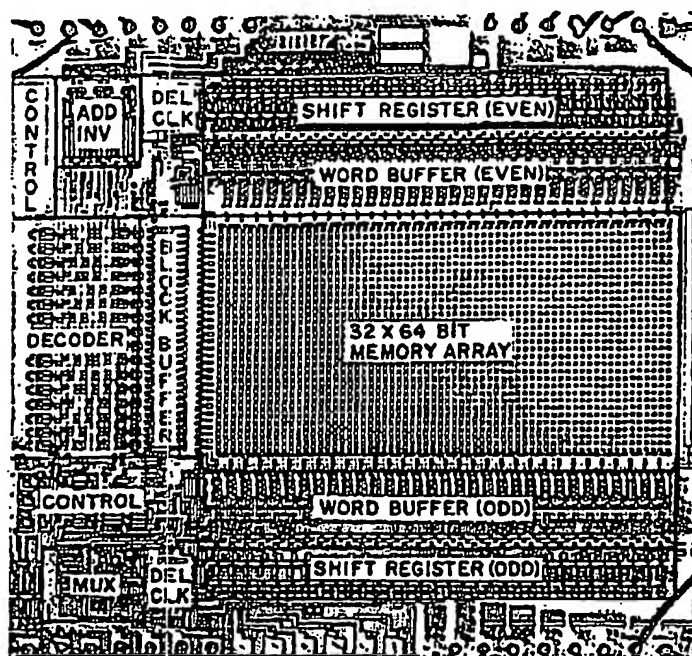| Organization	64K x 4 DRAM, 256 x 4 serial port
Package	24 pin, 400 mil DIP
Technology	LDD NMOS, 2 poly, 2 metal
Design rules	1.5μm
Cell size	4.5μm x 12μm
Chip size	173 mils x 361 mils
Row access time (Trac)	65ns
Serial access time (Tsca)	15ns
Active current	80mA (@ cycle times 175ns/25ns)
Standby current	5mA
Redundancy	4 rows and 16 columns

TABLE 1—Device characteristics.

FIGURE 1—196 x 196 mil² 2048-bit MNOS-BORAM chip
with functional blocks designated.

An I²L Watch Chip with Direct LED Drive
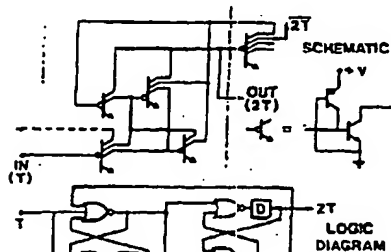
FIGURE 3—Divide by two circuit.
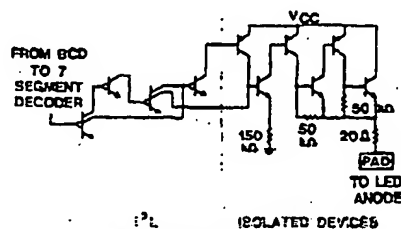
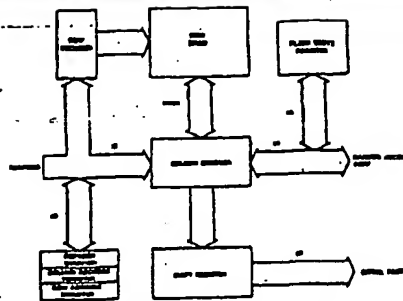FIGURE 4—Segment driver.
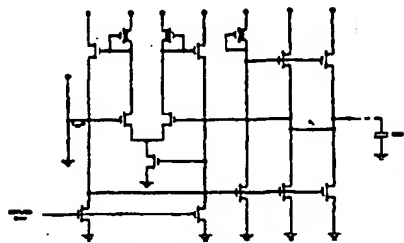
FIGURE 1—Video RAM architecture.

FIGURE 2—Schematic of cell plate driver.
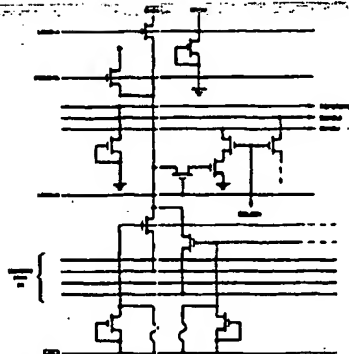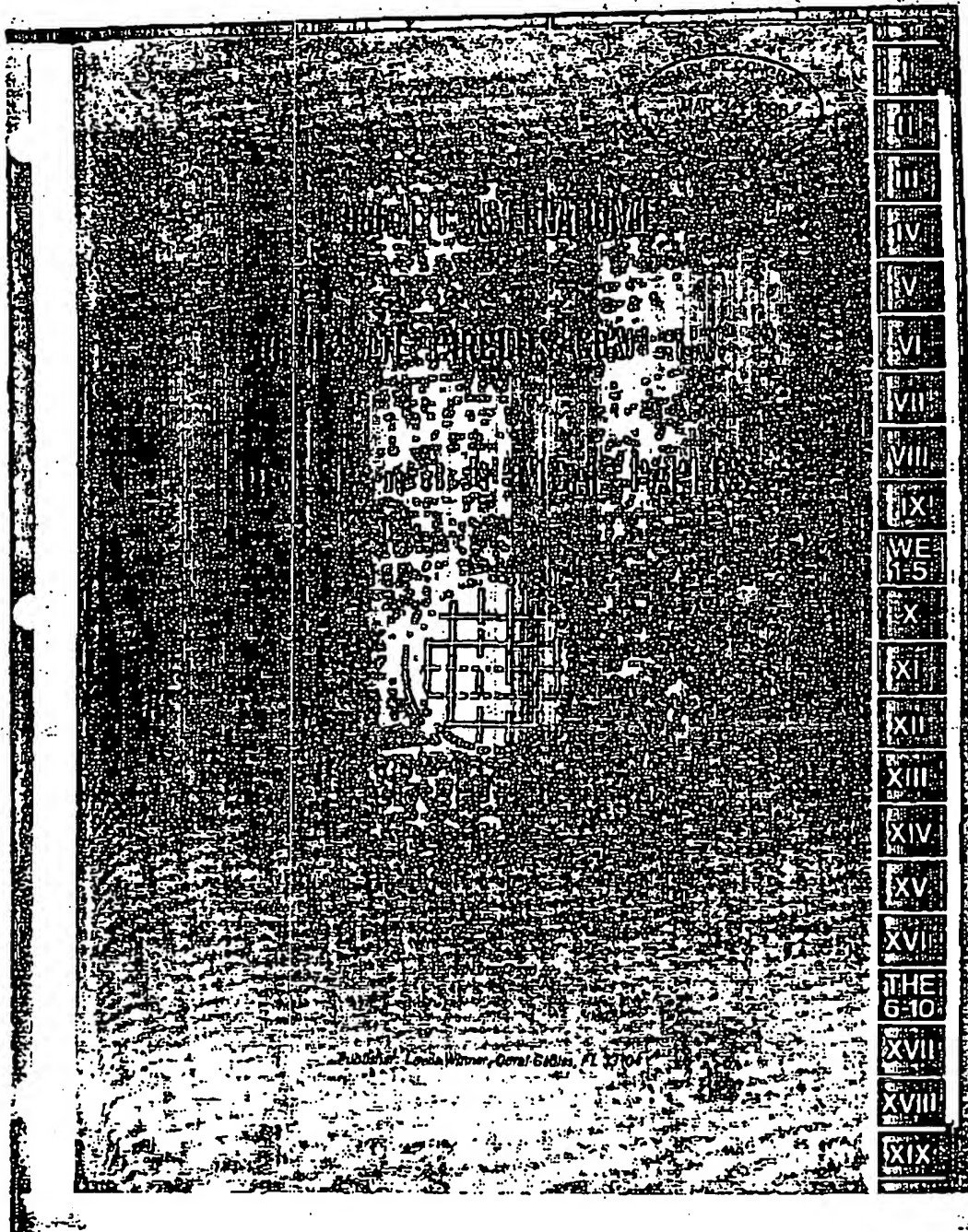

FIGURE 3—Schematic of video register and redundancy logic.

FIGURE 4—Serial clock input (top) and serial data output.
$V_{CC} = 5V$, 25°C.

1986 IEEE International Solid-State Circuits Conference

DIGEST OF TECHNICAL PAPERS

PRINTED IN THE UNITED STATES OF AMERICA

Volume XXIX

Regular Papers

A High Speed Dual Port Memory with Simultaneous Serial and Random Mode Access for Video Applications

RAYMOND PINKHAM, MEMBER, IEEE, DONALD J. REDWINE, MEMBER, IEEE, FRED A. VALENTE, TROY H. HERNDON, AND DANIEL F. ANDERSON

Abstract —A 64K x 1 NMOS dynamic RAM which is interfaced to an on-chip 256 bit high speed shift register is described. The device allows parallel transfer of 256 bits from a selected row in memory to the shift register in a normal RAS cycle time. Subsequently, the device provides simultaneous and asynchronous access from both the DRAM and the serial ports. The shift register can operate at a typical frequency of 33 MHz. When used in conjunction with multiple devices of the same design, a high resolution bit-mapped video display system can be achieved with video bandwidths beyond 100 MHz. The dual ported nature of the device allows a graphics processor to operate on the DRAM portion of the device while the shift register simultaneously provides a video data stream to a video display system.

I. INTRODUCTION

THE demand for color, combined text and graphics on a single screen, higher resolution displays, and real-time graphic simulators has fueled a growing trend toward bit-mapped graphics display systems where each pixel on the screen can be individually controlled by one or more bits of information in a bit-mapped memory. This technique provides unlimited flexibility in the images which can be displayed. Such memory intensive systems have been difficult to implement due to the lack of suitable memory devices which provide the density and the bandwidth necessary to supply information to the video to refresh the screen while also allowing a graphics processor sufficient access to the memory to update it and thus alter the image on the display. This paper describes a new memory device, designated the Multiport Video Memory, which combines a 64K x 1 dynamic RAM on the same chip with a high speed 256 bit shift register. A row of information in the DRAM is transferred to the register in a single memory cycle and is shifted serially out to the video display by a separate clock signal applied to the device. The dual ported nature of the device allows the DRAM

and the shift register to operate simultaneously and asynchronously. The DRAM array can be read from or written into while data are shifted serially into or out of the shift register. In a typical graphics system this provides a two- to three-fold increase in available bandwidth between the processor and the memory while significantly reducing the memory system chip count. The shift register can operate at a typical speed of 33 MHz and can be combined in parallel with other chips to provide video bandwidths in excess of 100 MHz.

II. BASIC ARCHITECTURE OF THE MULTIPORT VIDEO MEMORY

Fig. 1 illustrates a simple block diagram of the Multiport Video Memory, showing a 64K x 1 DRAM array connected to a 256 bit shift register. The Multiport Video Memory has two basic operating modes: asynchronous and transfer. In asynchronous mode, the DRAM and the shift register operate independently. Accordingly, $\overline{RAS}$, $\overline{CAS}$, D, Q, $R/\overline{W}$, and the multiplexed address pins (A_0 through A_7) serve the same functions as the respective pins of a standard 64K x 1 DRAM. In transfer mode, $\overline{RAS}$, $\overline{CAS}$, $R/\overline{W}$, and the addresses provide control and address information to allow 256 bits of information to be transferred in parallel from a selected row of the DRAM to the shift register or vice versa. Transfers are arbitrarily referenced to the DRAM array. Hence, transfers from the DRAM to the shift register are termed transfer reads and transfers from the shift register to the DRAM are termed transfer writes.

The transfer/output enable ($\overline{TR}/\overline{QE}$) pin has two functions. First, it controls whether the DRAM and the shift register will operate in transfer mode or asynchronous mode. Second, during asynchronous mode, it serves as an enable for the normal DRAM output to allow triple multiplexing of address, data in (D), and data out (Q) on a
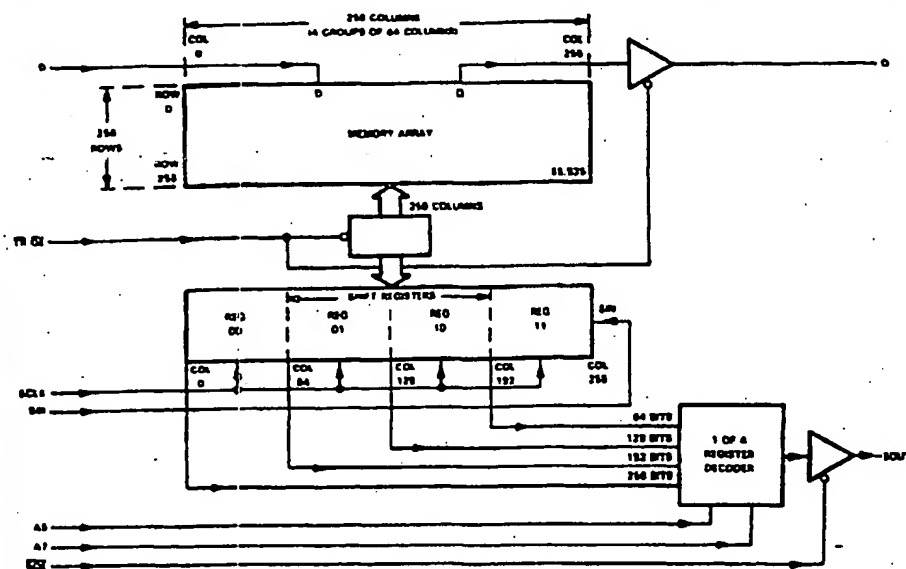
Fig. 1   Basic block diagram of the Multiport Video Memory showing
the DRAM array interfaced to an on-chip 256-bit shift register and the
control signals required to operate the random and serial circuitry.

common bus. The $R/\overline{W}$ pin also serves two functions. In asynchronous mode, it controls whether the DRAM will be read from or written to, the same as for a standard DRAM. In transfer mode, it controls whether 256 bits of data will be transferred from a row of memory to the register or vice versa.

In asynchronous mode, the serial input (SIN) and serial output (SOUT) pins shift data into and out of the shift register, respectively, under the control of the shift clock (SCLK) pin. The SIN pin provides added functionality for video and nonvideo applications. Registers from several Multiport Video Memory chips can, for example, be cascaded SOUT to SIN. Also, the DRAM array can be cleared quickly by shifting 256 zeros through SIN and performing a transfer write operation to each of the 256 rows of the array. This feature allows the memory to be cleared or "row patterned" in 70 μs instead of the 17 ms required for a standard 64K DRAM assuming a 260 ns cycle time.

The serial output enable pin ($\overline{SOE}$) is included to allow the SOUT to be tied into a bus shared with another bank of video memories or other video sources. When $\overline{SOE}$ is high, SOUT is in the high impedance state, freeing the bus for access by another source. Taking $\overline{SOE}$ low allows data to be shifted out normally.

The shift register is divided into four cascaded 64-bit segments as shown. A 2-bit binary code supplied by the

two most significant column addresses selects which segment is connected to the SOUT. This will be explained in Section IV.

III.   DETAILED DESCRIPTION OF THE DEVICE

Fig. 2 is a detailed block diagram showing the various internal control and clock signals of the Multiport Video Memory, while Fig. 3 shows a timing diagram of a transfer read followed by asynchronous DRAM write and serial shift operations. On the falling edge of $\overline{RAS}$, the row addresses, $\overline{TR/QE}$, and $R/\overline{W}$ are latched into input buffers by the row clocks. The row clocks generate control signals which sequence the row input buffers, the row decoders, and the transfer logic. The outputs of the $\overline{TR/QE}$, and $\overline{RAS}$ controlled $R/\overline{W}$ buffers are inputs to the transfer logic which sequences the word line and transfer line during transfer operations. On a transfer read cycle the word line is clocked high, allowing data from the row being transferred to be sensed and latched by the sense amplifiers. The transfer line is then clocked high, allowing data to be written into the shift register. For a transfer write operation, the sequencing of the word line and transfer line is reversed. Also, during either transfer operation, the two most significant column addresses may be strobed into pseudostatic latches on the falling edge of $\overline{CAS}$. The outputs of the latches are then decoded to determine which of
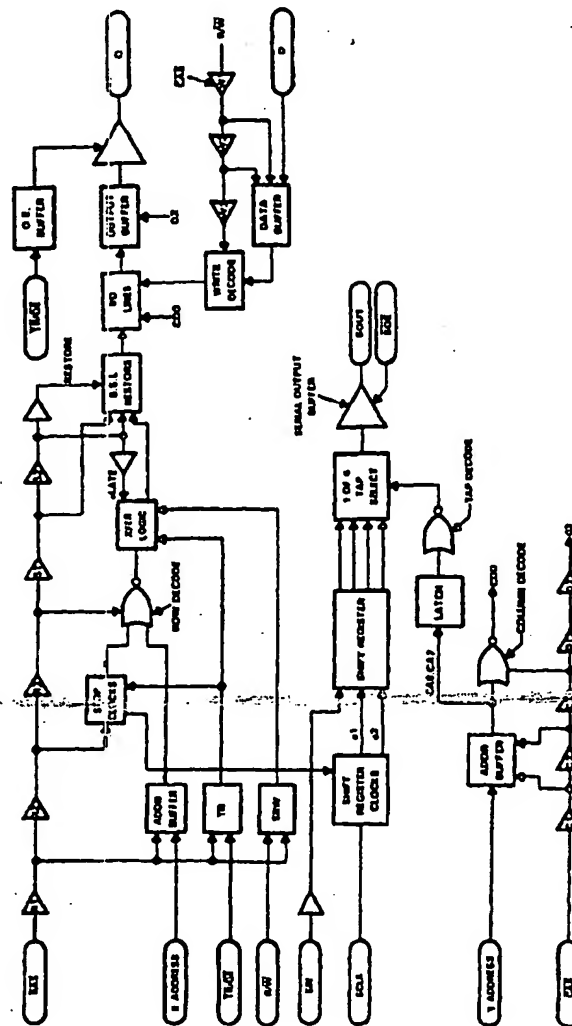
Fig. 2 Detailed block diagram of the device showing the internal clock scheme and circuit blocks. The DRAM circuits are interfaced to the serial and register transfer circuits via the transfer logic, line clocks and stop clocks, which are under the control of the $\overline{RAS}$, $R/\overline{F}$, and $\overline{TR}/\overline{QE}$ control inputs.
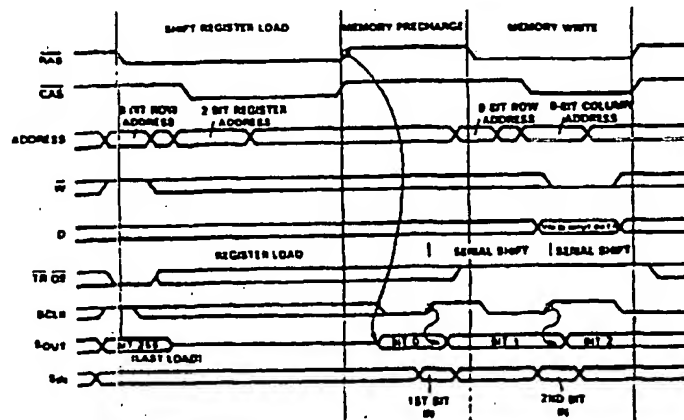
Fig. 3   Timing diagram showing a transfer read operation followed by
simultaneous and asynchronous DRAM write and serial shift cycles.
256 bits of information are transferred from a row in memory to the
shift register and are shifted out serially. The first bit out of the register
is triggered off the rising edge of RAS. All subsequent bits are propa-
gated through and out of the register by the SCLK input.

the four segments or "tap" points the register will begin
reading from. The first bit from the register is triggered
when RAS rises to complete a transfer cycle. Subsequently,
control is given back to the SCLK which triggers the
remaining bits out of the shift register. In transfer mode
(TR/QE low when RAS falls) the R/W signal is latched
on the falling edge of RAS by the row clocks, and the
DRAM write clocks are internally defeated. During
asynchronous mode, the DRAM write clocks are under the
control of the R/W and CAS inputs, with timing identical
to a standard DRAM, while the outputs of the RAS
controlled R/W and TR buffers connect normal DRAM
row clocks to the row and sense circuitry via the transfer
logic.

To perform transfer operations, additional circuit blocks
were inserted within the normal DRAM clock chains: the
stop clocks, the late clocks, and the previously mentioned
transfer logic.

The stop clocks are a series of delay stages, the purpose
of which is to ensure that all serial shifting of the register
has been completed before a transfer operation may take
place. These clocks are physically located within the row
clock chain of the DRAM, just ahead of the circuitry used
to generate the word line signal. During asynchronous
operation of the Multiport Video Memory these clocks are
essentially removed from the row chain by the use of a
shunt gated by internal transfer signals. During a transfer
cycle, however, the stop clocks are fully engaged and allow
sufficient delay and interlocking with shift register clocks
to ensure that the word line and transfer line signals are
disabled until the register has stopped shifting data. The
stop clocks are also used to generate reset signals for the
shift register clocks.
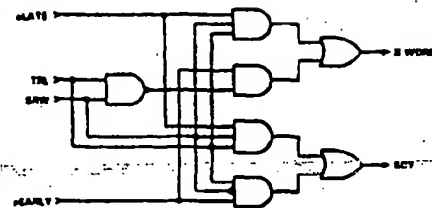
TR	R/W	X WORD	SCT	OPERATION
0	0	EARLY	LOW	DRAM READ
0	1	EARLY	LOW	DRAM WRITE
1	0	EARLY	LATE	TRANSFER READ
1	1	LATE	EARLY	TRANSFER WRITE

Fig. 4   Logic diagram of the transfer logic. During transfer read cycles,
the early clock is connected to the selected word line and the late clock
is connected to the register bit transfer line after sensing. During
transfer write cycles, the connections are reversed.

The late clocks are essentially a repetition of the clocks
used in the DRAM to generate the word line signal. This
circuit block is physically located several delay stages be-
yond the DRAM word line signal generator. The asynch-
ronous DRAM word line clock is also referred to as the
early clock. The late clock is only used during transfer
cycles to allow the data being transferred to reach their
destination.

The circuitry used to connect the early and late clocks
onto the word line and transfer line is the transfer logic.
The logic diagram for the transfer logic circuitry is shown
in Fig. 4. During a transfer read cycle, the TRL signal is
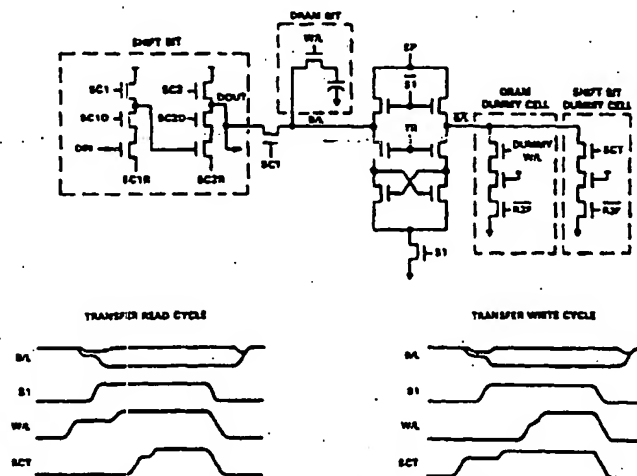latched high and the SRW signal is latched low. These

Fig. 5. (a) Circuit diagram of the DRAM storage cell, dummy cell, and bit lines, along with the register cell and the sense amplifier and restore circuits. (b) The internal timing waveforms for transfer read and transfer write cycles, respectively.

signals are generated by the $\overline{TR/QE}$ and $R/\overline{W}$ inputs. With the input signals to the transfer logic in this state, the early clock will go to the word line and the late clock to the transfer line (SCT). During a transfer write cycle, TRL is held high and SRW is held high. This allows the early clock to propagate onto the transfer line and the late clock to propagate onto the word line. The internal timing waveforms for transfer read and transfer write are shown in Fig. 5 along with the cell, word line, and sense circuitry. In normal DRAM operation the TRL signal is held low. This allows the early clock to drive the word line and also holds the transfer line low, thus disconnecting the shift register from the DRAM.

IV. SEGMENTED REGISTER ARCHITECTURE

Fig. 6 shows the physical composition of the shift register and the DRAM array in relation to the control blocks and the sense amplifiers. Register segment decoders with static segment address latches are also illustrated. As can be seen from the diagram, the serial data input (SIN) is first demultiplexed into one of two 128 bit shift registers on either side of the array and then multiplexed together at the output. Interleaving these even and odd bit registers allows one shift bit to be laid out in two column pitches and allows each of the shift registers to run at half the SCLK frequency.

The 256 bit shift register is actually composed of four cascaded 64 bit shift register segments. A register segment is implemented as two 32 bit register sections that are interleaved in and out of the device at the SIN and SOUT

pins, respectively. Each segment or "tap" selection is controlled by a 2 bit code applied as $\overline{CAS}$ falls to the two most significant column address pins during a shift register transfer cycle. If the 2 bits are 00, all 256 bits may be shifted out, starting at bit 00. A binary 01 permits 192 bits, starting at bit 64, to be shifted out. A binary 10 permits 128 bits to be shifted, starting with bit 128 of the 256 bit total. A binary 11 selects the last 64 bits, starting with bit number 192. The least significant bit with respect to the random access column address is shifted out first.

This segmented register architecture has advantages in two applications. First, it is useful in interlaced display systems where alternate horizontal lines are displayed on every vertical scan. Second, it is valuable in display systems where the number of pixels on a horizontal line is not a multiple of 256 bits. In both cases, unused bits can be passed over by chaining one of the internal tap points, allowing immediate access to the desired bits without having to shift the register until those bits appear at the 00 position of the 256 bit shift register. Thus, unwanted pixels are skipped over with minimum loss in time.

V. HIGH NOISE IMMUNITY DYNAMIC SHIFT REGISTER

The shift register bit of the Multiport Video Memory is shown in Fig. 7 along with its clock timing. This bit, a modified four phase six transistor shift bit using unconditional precharging [1], results in good noise margins and full dynamic operation over a wide range of voltage and temperature. In addition, the generation of the clock tim-
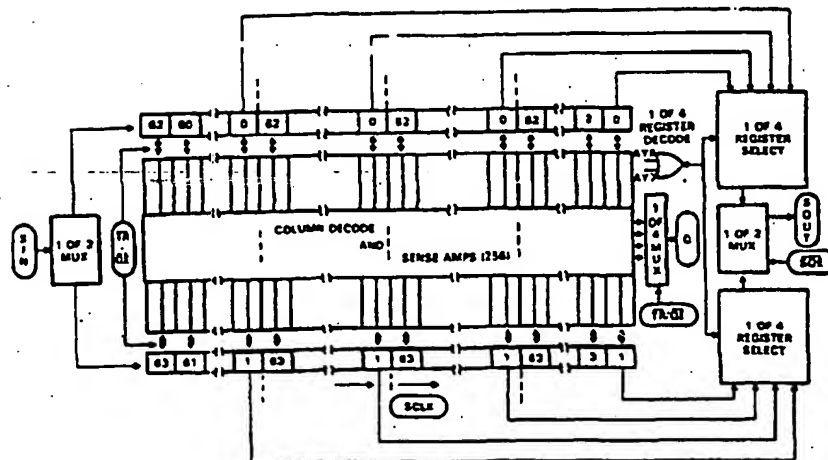
Fig. 6. Diagram of the Multiport Video Memory architecture showing the physical composition of the shift register and the function of the control signals. The register is implemented as four cascaded register segments with each segment divided into two sections, which are interleaved into and out of the register at the SIN and SOUT pins, respectively. Each section operates at half the device operating frequency.
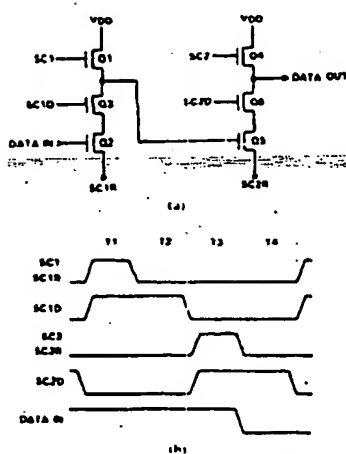

Fig. 7. (a) The four phase six transistor shift register cell designed for high speed and high noise immunity (b) along with the associated control clock waveforms.

ing is relatively straightforward. The return lines SC_1R and SC_2R play a crucial role in the operation of the shift register because they provide a means of determining if the data nodes within the register (those storing a logic zero) have been completely discharged. They also prevent leakage

through their respective driver transistors Q_2 and Q_5 when data = SC_1R = high by raising the threshold voltage of the driver transistors via the increased source to substrate voltage. These signals become particularly important on a transfer cycle when the shift register must be stopped without loss of data.

Each cycle for the shift register can be partitioned into four basic timing intervals as shown in Fig. 7; an unconditional precharge (T_1) followed by a data evaluation time (T_2) and a second unconditional precharge (T_3) followed by another data evaluation time (T_4).

Several features were added to the Multiport Video Memory to ensure proper timing and control. One feature makes use of SC_1R and SC_2R during a transfer read or transfer write cycle to allow asynchronous serial shift cycles which overlapped into the transfer cycle to be completed and to disable SCLK before the shift register is internally connected to the bit lines of the memory array. To accomplish this, an internal transfer signal is generated during the first portion of the transfer cycle which takes control away from the SCLK input and internally forces it to a low state. This ensures that both SC_1R and SC_2R are connected to ground potential after some propagation period. At this time, data evaluation in the shift register is complete because all charge has been removed from those storage nodes that are being discharged to logic zero. A voltage sensor circuit detects when both SC_1R and SC_2R have gone low, after which the stop clocks are activated and the early clock is enabled, connecting the bit lines to the DRAM cells (transfer read) or the shift register cells (transfer write).
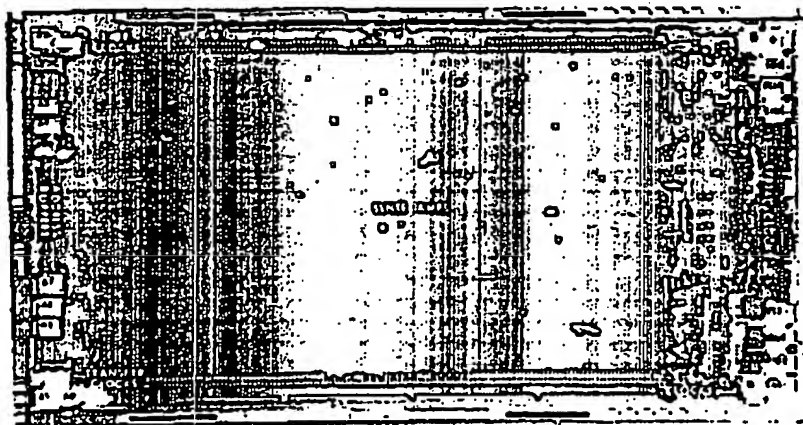
Fig. 8. A die photo of the Multiport Video Memory. The odd and even bit shift register sections are shown at the bottom and top of the DRAM array, respectively. Guard ring structures isolate the shift register sections from the memory array during asynchronous operation.

To maintain data integrity and to keep power in the register to a minimum, certain timing conditions must be met. The first condition is that SC_1D and SC_2D should not overlap since this would destroy data throughout the register. The second condition is that SC_1 and SC'_2 be turned off before SC_1R and SC_2R, respectively, are pulled to ground; failing to do so would result in a dc power path for all inverters with a "high" data state at their input. In order to guarantee the first of these timing conditions, the shift register clocks are interlocked so that overlaps between SC_1D and SC_2D cannot occur. An interlock also controls the timing between SC'_1 and SC_1R such that a dc path cannot be established in the shift register, likewise for SC'_2 and SC_2R.

VI. DEVICE TECHNOLOGY AND DESIGN FEATURES

The Multiport Video Memory has been fabricated with the SMOS (scaled NMOS) process. The process uses double level polysilicon with 300 Å oxides in the storage capacitor and the transistors in the periphery. The word line transfer gates are manufactured with 600 Å gate oxides. The four micrometer feature sizes yield a die size of 27.8 mm². The chip's dimensions and design rules pose no unusual manufacturing difficulties. Minimum polysilicon line width and spacing are 2.0 µm. Table I highlights the important process and design rule information. Fig. 8 is a die photo showing the various serial and DRAM circuit blocks.

Epitaxial silicon is incorporated to suppress potential noise generated in the substrate by the 33 MHz clock operation. Epi has been proven effective in controlling substrate noise and minority carrier injection in dynamic

TABLE I
PROCESS TECHNOLOGY HIGHLIGHT INFORMATION OF THE MULTIPORT VIDEO MEMORY

TECHNOLOGY	SCALED NMOS (SMOS) DLP
102 (STORAGE CELL PERIPHERY)	300 Å
CELL SIZE	8.33 µm 19.3 µm
CHIP SIZE	27.8 mm²
MIN DESIGN RULE	3.5 µm
CHANNEL LENGTH	2.7 µm

storage arrays [2]. It also decreases p-n junction and field induced leakage, which improves device refresh times [3]-[6]. The shift register and array are separated by approximately 178 µm and within this space an n⁺ diffused guard ring is placed that is biased to +5 V. With +5 V applied to the n⁺ diffusion, an electric field exists extending 2-3 µm into the silicon [7]. Minority carriers injected into the epi layer near the shift register have negligible probability of drifting along the narrow gap within the electric field between the n⁺ guard ring and the p-p interface.

The electrostatic discharge (ESD) protection structures on the Multiport Video Memory input and output pins have achieved the most effective ESD protection reported to date for a MOS memory. Failure thresholds of greater than 7 kV by MIL-STD 883B method 3015.1 have been demonstrated on all pins. The extremely high failure threshold of the protection devices are the result of experimental studies which identified the failure mechanisms of ESD structures and determined the most effective circuit network [8]. The study revealed that not only is the circuit network important, but layout technique is extremely critical. Design guidelines established by this study were used to design the ESD structures on the Multiport Video
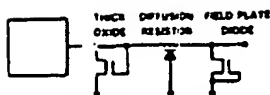
Fig. 9. A circuit schematic of the electrostatic discharge protection circuit used on the device pins. Layout of the devices is a critical part of the design. Failure threshold voltages above 7 kV have been achieved.

TABLE II
TYPICAL DEVICE PERFORMANCE CHARACTERISTICS OF THE MULTIPORT VIDEO MEMORY
THE TEMPERATURE IS 25°C AND THE SUPPLY VOLTAGE V_{DD} IS 5.0 V

PACKAGE		20 Pin 300 mil DIP	
POWER SUPPLY		SINGLE 5 V ± 10%	
ACCESS TIME	tRAC	150 ns	
	tCAC	70 ns	
POWER		DRAM + SR	DRAM ONLY
	ACTIVE	45 mW	25 mW
	STANDBY	8 mW	4 mW
PERIOD		256 CYCLE 4 ms	
SR CLOCK RATE	MAX	33 MHz	
	MIN	1 kHz	

Fig. 10. Oscillographs of device operation. (a) Transfer read followed by asynchronous operation and (b) asynchronous operation showing the 33 MHz operation of the serial shift out.

Memory. The ESD protection on the input pins is a two stage circuit consisting of a thick field oxide metal gate resistor, a diffused resistor, diode, and a polygate field plate diode, as shown in Fig. 9. The output devices have a protection circuit whose layout structure consists of the straight poly fingers of the output transistors themselves, which follow the same guidelines used on the input ESD structures.

VII. DEVICE PERFORMANCE

Table II lists the typical measured performance data for the Multiport Video Memory. All DRAM performance data are typical of currently available 64K DRAM's with RAS access times of 150 ns. In addition, data are included for serial operation and operating power with and without the shift register operating. Fig. 10 is an oscillograph showing the actual voltage waveforms diagrammed in Fig. 3. The SCLK is defeated internally during the transfer read and has no effect on the SOUT. RAS going high triggers the first bit out of the register after which control is returned to the SCLK. Also shown in Fig. 10 is an oscillograph of the SOUT and serial SCLK showing the 33 MHz operation.

VIII. CONCLUSIONS

A high speed dynamic random access memory device has been developed which interfaces to an on-chip 256 bit shift register and contains all necessary control signals to transfer data from the shift register to the 256 columns of a single memory row or vice versa. The shift register can operate simultaneously and asynchronously with respect to normal DRAM access and can achieve a typical maximum shift frequency of 33 MHz. This allows the device to deliver a high speed serial data stream to, say, a video system while allowing a processor to alter the contents of the memory array. The shift register is segmented into four
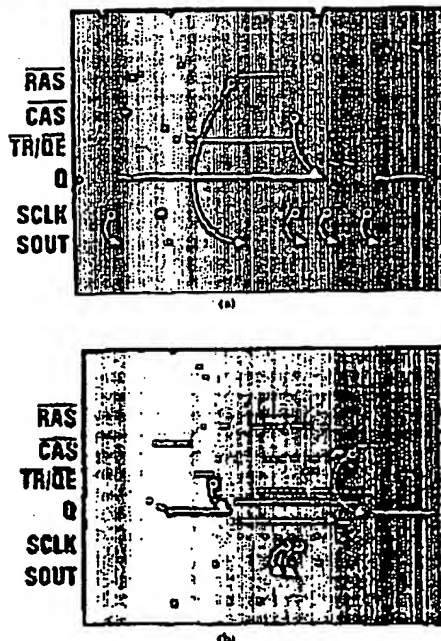
cascaded sections, and a decoder circuit is provided to allow the register to be read out from any one of four tap points along the register, which are located on 64 bit boundaries. The register and associated clock circuits are designed for fast operating frequencies and high noise immunity. While using fairly conventional process technology, epitaxial silicon is employed for noise suppression and control of minority carrier injection. The protection achieved against electrostatic discharge exceeds 7000 V on all pins. Novel process tolerant sampling and modeling techniques were incorporated to improve the accuracy of simulations and enhance yield.

REFERENCES

[1] W. N. Carr and J. P. Mize, "MOS/LSI design and applications," in *Texas Instruments Electronics Series*. New York: McGraw-Hill, pp. 156–157.

[2] J. G. Posa, "Is epitaxy right for MOS?," *Electronics*, Feb. 10, 1981.

[3] J. W. Slotboom *et al.*, "Leakage currents in high density CCD memory structures," *IEDM Dig.*, 1981, pp. 667–670.

[4] B. Eitan, D. F. Bentchkowsky, and J. Shappir, "Holding time degradation in dynamic MOS RAM by injection-induced electron currents," *IEEE Trans. Electron Devices*, vol. ED-28, p. 1515, Dec. 1981.

[5] T. Furoyama, K. Ohuchi, and S. Kohyama, "An electrical mechanism for holding time degradation in dynamic RAM's," *IEEE Trans. Electron Devices*, vol. ED-26, p. 1694, Nov. 1979.

[6] S. Tam, F. C. Hsu, P. K. Ko, C. Hu, and R. S. Muller, "Hot electron induced excess carriers in MOSFET's," *IEEE Electron Device Lett.*, p. 376, Dec. 1982.

[7] A. S. Grove, *Physics and Technology of Semiconductor Devices*. New York: Wiley, p. 163.

[8] C. Duvvury, R. N. Rountree, and L. S. White, "A summary of most effective electrostatic discharge protection circuits for MOS memories and their observed failure modes," in *Elev. Overstress Electrostatic Discharge Symp. Proc.*, Sept. 1983.

Raymond Pinkham (S'77-M'78) was born in Chicago, IL, in 1956. He received the B.S. degree in electrical engineering from the University of Illinois, Urbana, in 1978 and did graduate work at the University of Houston, Houston, TX.

In 1978 he joined the MOS Memory Design Group, Texas Instruments, Inc., Houston, TX, where he worked in the design and development of high speed static RAM's until 1982. Since then he has worked as Design Manager for high speed dual access multiport memories with an emphasis on graphics applications. His current interests are in computer graphics and the scaling and modeling of VLSI NMOS and CMOS technologies.

Mr. Pinkham is a member of Eta Kappa Nu, the IEEE Electron Devices Society and the IEEE Computer Society.

Donald J. Redwine (S'62-M'78) was born in Houston, TX, on June 6, 1940. He received the B.S. degree in electrical engineering from Texas A & M University, College Station, TX, in 1963 and did graduate work at the University of Texas, Austin.

He joined Texas Instruments, Inc., Houston, TX, in September 1959 and is presently a Senior Member of the Technical Staff. His design experience has been in ECL, CCD, and MOS technologies, and he is presently employed in the MOS memory DRAM activity.

Mr. Redwine holds 14 patents related to circuit design and is a member of Eta Kappa Nu.
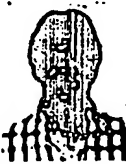
Fred A. Valente was born in New York, NY, on September 30, 1957. He received the B.S. degree in electrical engineering from Manhattan College, Bronx, NY, in 1979.

He joined the MOS Memory Division, Texas Instruments, Inc., Houston, TX, in May 1979. Since then he has worked as a Design Engineer on high and medium performance static RAM's and is presently a Design Engineer on the high speed multiport video memory project in the DRAM Product Group.

Troy H. Herndon was born in Dallas, TX, on October 1, 1942. He received his education at the University of Texas, Arlington, and at military technical schools.

He joined Texas Instruments, Inc., Houston, TX, in 1966 and worked in the MOS Section, Semiconductor Research and Development Laboratory. Since then he has been associated with MOS military programs, CMOS, design automation, and high speed MOS static RAM design. He is now working on the design of high speed multiport memories within the DRAM Product Group. His interests are in VLSI chip layout and CAD/CAM development.

Daniel F. Anderson was born in Silver City, NM, on June 21, 1954. He received the Certificate in Electronics Technology from Albuquerque Technical Vocational Institute in 1979.

He joined the MOS Memory Division, Texas Instruments, Inc., Houston, TX, in September 1979 where he worked as a Design Technician in the high speed static RAM area until 1982. Since then he has worked in the design of high speed multiport memories within the DRAM Product Group.

81

SESSION III: SPECIAL APPLICATION MEMORIES

Chairman: Dennis Segers

Mostek Corp.

Carrolton, TX

WAM 3.1: A 256K Dual Port Memory

Syoji Ishimoto, Akira Nagami, Hiroshi Watanabe, Junji Kiyono,

Noboru Hirakawa, Yasushi Okuyama

NEC Corp.

Kawasaki, Japan

Fumio Hosokawa, Kazuo Tokushige

NEC IC Microcomputer Systems, Ltd.

Kawasaki, Japan

THIS PAPER WILL COVER a 256K dual port memory, a 64K x 4 DRAM plus a virtually asynchronous 256 x 4 serial readout memory.

Although a memory of this type has already been described[1], this memory has features that make it suitable for advanced graphic applications, as outlined below.

The data from the serial port can be readout continuously, even when the data are being transferred. This function improves transfer efficiency to 100% for any display size.

The serial readout can be started and stopped at any location among 0th to 255th without spending idle time. This function makes vertical scroll, for example, easier by only changing start address. Furthermore, even a fraction of a full image pattern can be displayed as a continuous data flow by the combination of the real time data transfer and this pointer function. Additionally, the write-per-bit function allows partial write to some of 4b. Thus, it is possible to change colors without read modify write.

A simplified block diagram of the memory is shown in Figure 1. The block is divided in two portions. One is a 64K x 4 RAM port, the other is a 256 x 4 serial read port. The RAM port is a conventional DRAM, except for the write-per-bit control. The serial port consists of 1024-bit data registers and a one-out-of 256 multiplexer, that allows start of data transfer at any location.

Figure 2 shows timing diagrams relating data transfer. Both ports are fully asynchronous except when a transfer from the DRAM cells to the data registers is being executed. The input DT (Data Transfer) selects a RAM or Data Transfer cycle. A cycle started on the condition of DT High is a normal RAM operation, and a DT Low cycle is a Data Transfer cycle (DT cycle). During DT cycle, the RAM port cannot be accessed. However, the serial readout can be executed in any DT cycle to keep fully continuous data flow. A detailed timing diagram of a DT cycle is shown in Figure 3. When a DT cycle is requested, row address inputs define a new row line to be transferred to the data registers, and then the subsequent column address is acknowledged as a starting location of 256 serial data and stored in the start address latch. Even if this DT cycle is

[1] Pinkham, R., et. al., "Video RAM Excel at Fast Graphics", Electronic Design, p. 161-171; Aug., 1983.

started, the serial port can read the old row line data. When the DT is turned to High, it transfers the new row line data to the serial data registers, and then enables the readout of the new serial data from the location indicated by the column address. It should be noted that these realtime data transfer and pointer functions make a fraction of a image pattern as a fully continuous data flow.

An output circuit has been introduced to achieve this high speed realtime data transfer. The serial readout data can continue even during the data transfer cycle. A temporary 4b wide data register is loaded with the first four parallel bits to be readout through the serial port, directly from the I/O bus of the DRAM. Thus, the register delivers the first data as early as in a normal serial read cycle. The next and succeeding bits will be read out from the 1024b data register.

Figure 4 shows a timing for write-per-bit operation in the DRAM. A normal write operation keeps the WB/WE high as RAS falls and then turns it to low. On the other hand, a write-per-bit operation keeps the WB/WE low as RAS falls, and write (high) or don't write (low) instruction is supplied separately on each data I/O line. Thus, no additional control pin is necessary.

Figure 5 shows a microphotograph of the dual port memory. Figure 6 is a photograph showing waveforms. The DRAM port and the serial port are operating asynchronously.

Table 1 shows summarized typical characteristics.

FIGURE 1—Block diagram of memory.

FIGURE 2—Timing diagram of total memory operation.
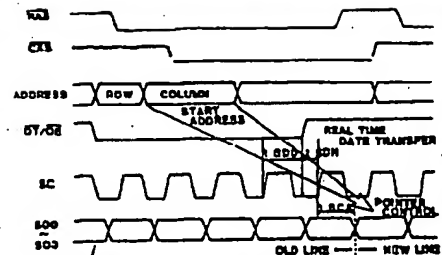
FIGURE 3—Timing diagram of the data transfer cycle.
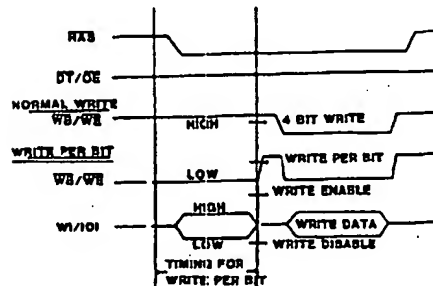
FIGURE 4—Write per bit in the RAM portion.
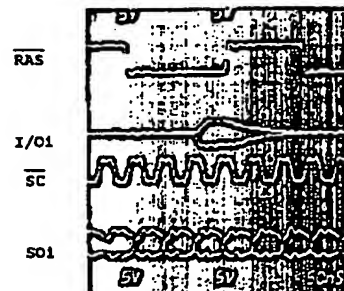
FIGURE 6—Asynchronous operating waveforms of memory.
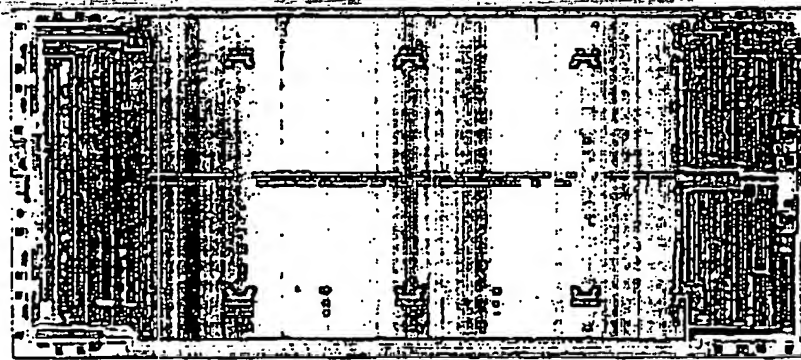
FIGURE 5—Chip photograph of memory.

1985 IEEE INTERNATIONAL

SOLID-STATE CIRCUITS CONFERENCE

DIGEST OF TECHNICAL PAPERS

February, 1985

New Orleans, LA

I
II
III
IV
V
VI
VII
VIII
IX
WE
1-6
X
XI
XII
XIII
XIV
XV
XVI
THE
7-11
XVII
XVIII
XIX

1985 IEEE International Solid-State Circuits Conference

DIGEST OF TECHNICAL PAPERS

TK 7870
.I57
vol. 28
1985

PRINTED IN THE UNITED STATES OF AMERICA

Volume XXVIII

DESIGN APPLICATIONS

Internally timed RAMs build fast writable control stores

Mohammad Shakaib Iqbal
Fujitsu Microelectronics Inc., 3545 N. First St., San Jose, CA 95134-1804; (408) 922-9000.

The increasing speed of mainframes and minicomputers produces a need for memory access even faster than that supplied by ECL RAMs. One way to cut into 15-ns memory-access times is through process improvements, but this avenue quickly reaches its limits. Another method is to rework the architecture of the writable control store, which holds the microinstructions that implement the machine's assembly-language instructions. For instance, adding registers in the address and data lines to the control memory causes a pipeline effect that speeds up both read and write operations.

But the number of registers needed to process the size of control words in some of today's minicomputers can be prohibitive. The solution lies in the new self-timed RAMs (STRAMs)—pipelined memory devices containing on-board registers or latches, as well as a write-pulse generator. STRAMs not only shrink access times to 7 ns, but they also cut board space and reduce the number of lengthy connections between discrete parts. The latter is important because at ECL speeds these leads act as transmission lines, generating reflections and crosstalk.

To better understand how a STRAM can help a designer perform a specific task, consider a minicomputer's basic architecture. Both mainframes and minicomputers use microprogrammed processors in their CPUs. A microprogram is a flexible way to generate the control signals that implement assembly language. These control sequences or microinstructions reside in a control memory, usually a set of PROMs addressed by a microprogram counter.

In a microprogrammable machine, however, the control memory consists of fast RAMs, so a user can alter the control signals and modify the instructions. For example, a typical minicomputer CPU

Create faster computers without sacrificing board space. Self-timed RAMs do the trick, replacing standard ECL RAMs in control memories.

1. In a typical microprogrammed CPU, a control unit holds a control word employed for register loading, identification, and reading.

contains 12 kbytes of microprogrammable memory in its writable control store to diagnose problems, perform certain instructions, and change the microcode. For the sophisticated user, the CPU has an extra 12 kbytes of writable control store. This architecture lets a user change the way the computer responds to machine-language instructions.

A microprogrammable CPU usually contains general-purpose registers, an instruction register, a memory data register, a memory address register, a program counter, a 16-function arithmetic logic unit, a temporary register called an accumulator, and a control unit (Fig. 1). The memory data register holds the data word to be sent to the memory, and the memory address register holds the address to the memory. The control unit sends a control word for register identification, loading, and reading. It generates signals like memory read and write, accumulator read and load, and ALU operations. The accumulator holds the ALU inputs and outputs.

The writable control store is implemented within the

2. Adding registers to a writable control store's data and address paths speeds up the computer but at a steep price in board space. An alternative is to replace the components in the highlighted area with a self-timed RAM, which contains a write-pulse generator and registers.

control unit (Fig. 2). Its task is to generate the correct sequence of steps to execute the assembly-language instruction. Included in the controller are a starting address generator, microprogram counter, control memory, and control register. The control memory, addressed by the microprogram counter, stores the microinstructions. The control register holds the control word.

The process begins when the CPU fetches a machine-language instruction from the main memory and loads it into the instruction register. Microprogramming then takes over. The instruction register puts the instruction into the starting address generator, which decodes the address of the first microinstruction in the control memory and loads this address into the microprogram counter. Next, the contents of the control memory pointed to by the microprogram counter are fetched and loaded into the control-word register. The microprogram counter is then updated to point to the next microinstruction in the desired sequence.

Minicomputers have control words 10 to 100 bits long. Each bit placed into the control-word register controls a part of the computer, including the instruction register, program counter, accumulator, memory, and ALU control. Hence, each bit is connected to a specific destination. The various control signals open or close data paths to these destinations or instruct the locations to perform an operation. For example, to transfer data between two registers, a control signal must instruct the source register to place the data on the bus, and a second signal must tell the destination register to read the data on the bus.

If the control store is writable, there must be a multiplexer between the microprogram counter and the control memory, because the address can come from either the microprogram counter or the system address bus. The system address bus's only task is to write to the control memory.

This is where a register between the counter and control memory input is beneficial. While the microprogram counter is generating an address during a read cycle (when it increments), the previous address can be in the register pointing to the control memory. That's the desired pipeline effect.

The computer gains a similar advantage during write cycles—that is, when the instructions in the microprogram are being altered. In this case, the new data is carried over the system bus and written in the control memory. If the memory consists of standard ECL RAMs and no registers, the address-hold time requirement will slow down the process.

Adding a register again creates a pipeline effect because the address and the data are both placed in the register. The address remains valid on the register's outputs until a new clock edge arrives, bringing a new address from the microprogram counter. The data and the address inputs are placed in the register on the true ongoing
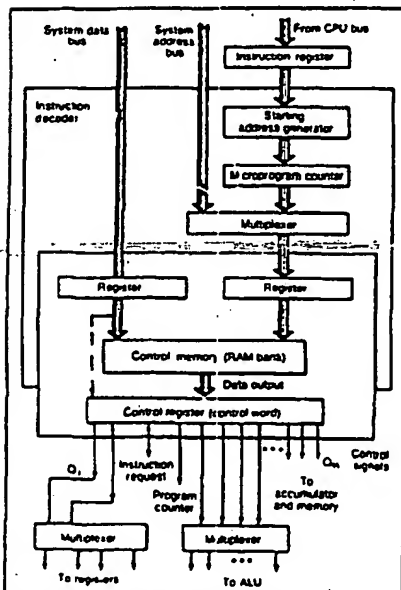
edge of the clock. The Write Enable signal is also placed in the register (Fig. 3a).

The several nanoseconds saved on each read and write cycle can add up to a considerable speed increase during normal computer operation. As noted, using STRAMs gives the designer this speed boost without the space penalty exacted by discrete registers.

In the example noted, a totally pipelined architecture was desired, so the registered STRAMs were used. This configuration yields the highest bit rate at the system level because the succeeding cycle can begin while the output signal is slewing and propagating. The data isn't available at the outputs until the next clock edge.

In some computers, however, the control store might have to read data from the RAM in one memory cycle. When this is the case, the control memory's inputs must have latches to hold the input data and address for saving the hold times. The output lines are also latched so that data can be placed on the data bus in one cycle. A latched STRAM fills the bill. This device's timing diagrams show that in read cycles the data is read in the same memory cycle (Fig. 3b).

In a STRAM, the Address, Data In, Chip Enable, and Write Enable signals are latched into the on-chip registers or latches by the true-going edge or level of the clock pulse at the start of the memory cycle. All these signals remain valid throughout the memory cycle until the next true-going clock edge or level. As a result, signals need not be held stable during the entire cycle. They can slew down during one cycle to prepare for the next one.

It's advantageous to trigger the write operation at the true going clock edge by latching the Address, Data, and Write Enable signals. Then the new Data and Address signals can be placed at the inputs while the old data is being written to the RAM cells. Also, this technique eliminates address skew because all the timing is clock-edge driven.

The basic difference between the registered and the latched STRAM, in fact, is that the former is clock-edge sensitive, while the latter is level sensitive (Fig. 4). During a registered STRAM's read cycle, the data is available in the next clock cycle. For the latched STRAM, the data is available during the same memory cycle.

An advantage of both the latched and the register STRAM, however, is the built-in write-pulse generator, which eliminates an annoying problem associated with
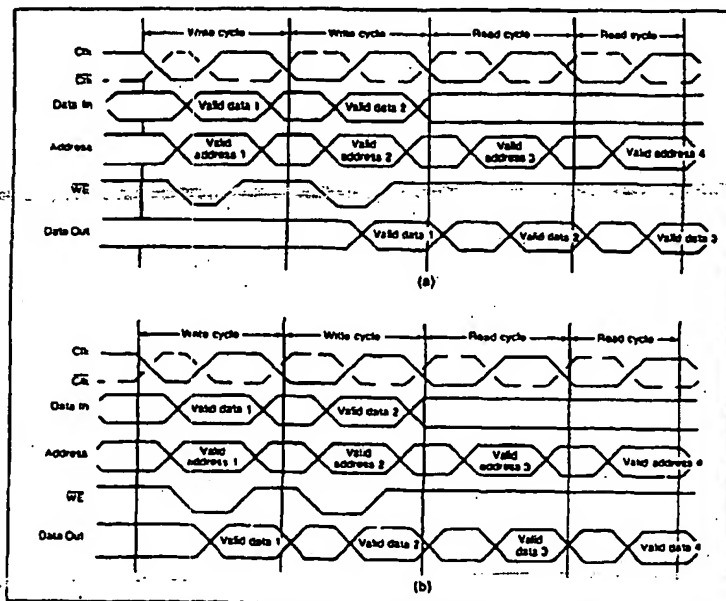
3. Timing diagrams show that in a registered STRAM (a) the control word is read in the second clock cycle, while a latched STRAM (b) reads the data in the same clock cycle.
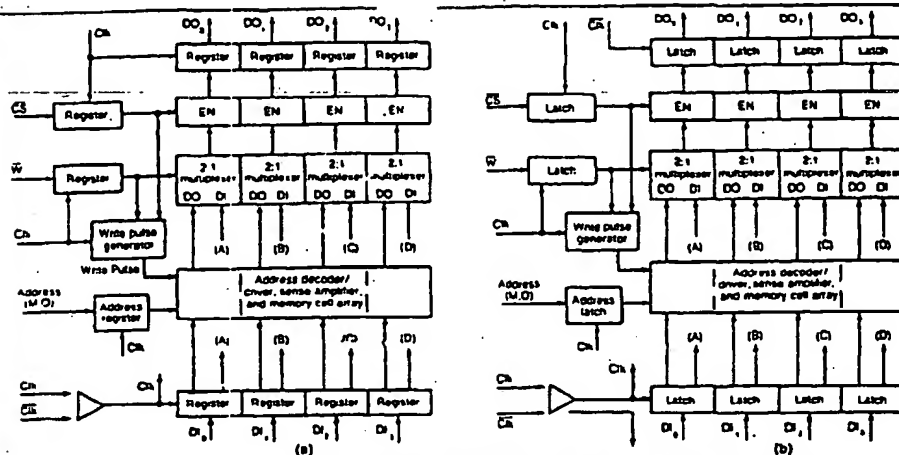
4. Both the registered (a) and latched versions (b) of the STRAM include a write-pulse generator. The devices have differential clock inputs—Clock and Clock—but single-ended operation is possible by connecting either clock line to an internal reference voltage.

fast ECL RAMs—the generation of a narrow write pulse. This on-board capability not only simplifies the designer's task, since creating very narrow pulses can be difficult, but it also speeds up the write cycle.

For instance, the length of a write cycle for a typical static RAM, MBM10474-15, employed without input and output latches is the sum of the minimum setup time, 2 ns; the write-pulse length, 12 ns; and the minimum hold time, 1 ns. That comes to 15 ns. For a latched STRAM with an internal write pulse generator, MBM10476LL-9, the write cycle time is the minimum setup time, 1 ns, plus the minimum high or low clock time, 6 ns—a total of 7 ns.

Another advantage of the STRAM is that the data written in the RAM is transparent to the outputs. This boosts the speed of the system for a cache write-through and improves the write-cycle timing for the writable control store. Also, the input data is transparent to the output in the same clock cycle for the latched STRAM and in the next cycle for the registered version. The transparent feature is helpful in diagnostic tasks and for writing back the data into the next location.

In both types of STRAMs the setup and hold times are identical for all inputs, simplifying the timing. The sum of the setup and hold times, also called the required valid window, is only 30% of the overall cycle time. For example, a 1k-by-4 latched STRAM, the MBM10476LL, has a clock cycle of 10 ns and a setup time plus hold time of 3 ns. This low ratio leaves enough time for the inputs to get ready for the next cycle.

The read and write cycles also have the same timing, because the data-input registers and latches are loaded at the start of each cycle, regardless of the type of cycle. This balanced read-write configuration is helpful for system integration. When Write Enable is low at the beginning of a cycle, an internal write operation writes the data into memory and restores internal write lines to their original values.

The devices have differential clock inputs—Clock and Clock—to increase timing accuracy. They can be connected in either the differential or single-ended mode. In the differential mode, data is latched at the cross point of the rising edge of Clock and at the falling edge of Clock. Connecting either Clock or Clock to the internal reference voltage configures the STRAM in the single-ended mode, latching data at the true going edge of the clock.□

Mohammad Shakaib Iqbal, an application engineering supervisor at Fujitsu Microelectronics Inc., works on local-area networks, microcontrollers, small computer system interfaces, and memory products. He holds a BSEE from NED University, in Karachi, Pakistan, and an MSEE from Oregon State University.

How Valuable?	Circle
Highly	541
Moderately	542
Slightly	543

720

A 0.5-GHz CMOS Digital RF Memory Chip

WILLIAM M. SCHNAITTER, EDWARD T. LEWIS, AND BRUCE E. GORDON

Abstract —Digital RF memories (DRFM's) are key elements for modern radar jamming. An RF signal is sampled, stored in random access memory (RAM), and later recreated from the stored data. Here we describe the first CMOS ($l_{eff} = 1$ μm) DRFM chip, interpreting static RAM, control circuitry, and two channels of shift registers, on a single chip. The sample rate achieved was 0.5 GHz. VLSI density was made possible by the low-power dissipation of quiescent CMOS circuits. An 8K RAM prototype chip has been built and tested.

I. INTRODUCTION

MODERN radar systems employ sophisticated anti-jamming techniques through signal processing of the outgoing and incoming pulses. To combat such radars, radio-frequency memory (RFM) has been employed in many electronic countermeasure (ECM) systems. The memory elements of such systems have taken many forms. More recently, digital RFM (DRFM) systems have been built using IC's with silicon static random access memory (RAM) as the memory element [1]. Through the ability to record any analog signal as a 1-bit (pulsewidth only) digital replica, in RAM much greater flexibility and longer data retention have been achieved. The signal may then be retransmitted, using a variety of post-processing techniques to effect the ECM. Currently, DRFM systems employ ECL-based shift registers and use off-chip RAM. These high-cost systems have generally been built from off-the-shelf packaged IC's and have been characterized by high part count. The high-power dissipation requires extraordinary cooling measures, such as liquid immersion. Recently, DRFM's have been predicted which would employ GaAs shift register IC's and even GaAs RAM's [2]. These would also lead to problems of high thermal dissipation, high part count, and high cost.

This paper describes the first CMOS ($l_{eff} = 1$ μm) DRFM chip, termed the fast digital memory chip or FDMC. The shift registers (clocked at 0.5 GHz) and RAM (8K) are on a single chip along with control circuitry and other special purpose functions. In contrast to other technologies, the low-power dissipation of quiescent CMOS circuits permits VLSI density. The 8K RAM prototype chip has been built and tested. With existing production technology, static memory capacity of 64K and higher is certainly feasible.

TABLE I
SOME PARAMETERS OF THE 1.25-μm PROCESS

	p-ch	n-ch
V_{th}	-1.09 V	2.07 V
T_{ox}	250 Å	250 Å
L_{eff}	1.1 um	1.1 um
Junct. depth	0.33 um	0.18 um
N_{sub}	$1.2E16$ cm^{-3}	$3E16$ cm^{-3}
contact size	1.3 um	
LOCOS thickness	0.52um	
epi thickness	5 um, as deposited	
P-well jn. depth	3.5 um	

A 1.25-μm process was used for wafer fabrication. Table I summarizes the design rules. In achieving these high speeds, the normal design rules were employed without the need to "push" the process, which would have been at the cost of reduced yield.

Considerations in the design of this chip, the final chip form, and the test results to date will be discussed.

II. OVERALL CHIP DESCRIPTION

Fig. 1 shows a view of the prototype chip. The chip has two basic operating modes: 1) as a serial I/O RF memory; and 2) as a normal RAM. One pin, called "M," switches the operation from one mode to the other. Fig. 2 shows a simplified block diagram of the chip.

In the RF or "serial" mode, the data input—a preprocessed sinusoidal signal with constant amplitude—is applied to the RF input, sampled, and clocked alternately into one of two 32-bit shift registers called SSR A and SSR B (signal shift register A or B; see Fig. 3). One of the SSR's is shifting and the other is frozen at all times. An SSR freezes to permit either or both of two functions: 1) storage of the shift register contents into the RAM portion of the chip; or 2) recall from the RAM of previously stored data (preconditioning of the shift register). It is possible to perform both functions (storage first) on one shift register within one freeze period. In this case, during the subsequent shift period, the 32 bits of recalled data are shifted
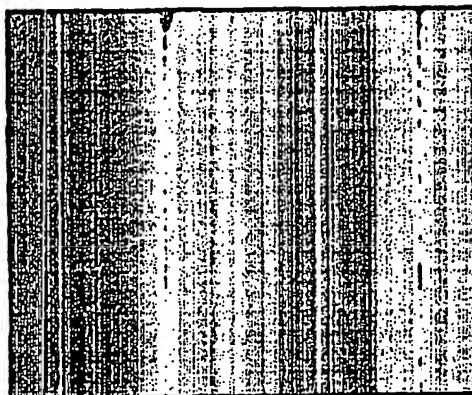
Fig. 1. Photograph of 1K RAM DRFM chip. Serial-in is at top and serial-out is at bottom. Chip measures 7100 × 3600 m².
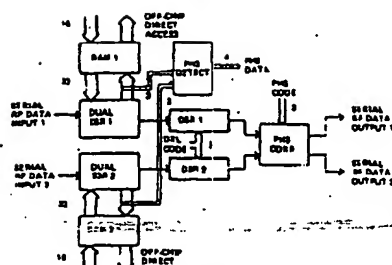
Fig. 2. Simplified block diagram.
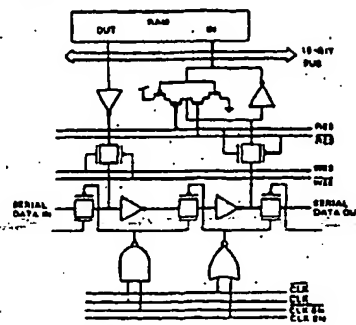
Fig. 3. One stage of the SSR.

out while 32 bits of new data are simultaneously shifted in, to be stored in the next freeze period. Thus, with the A/B alternation, the SSR pair acts as a continuous bidirectional serial–parallel converter. While the chip is in this operating mode, the RAM is organized in 32-bit words to match the SSR.

The sinusoidal RF input signal gets reduced to a digital representation, in this case one bit. All information regarding signal amplitude will be handled by other portions of any system in which the FDMC is used. Only the time-dependent frequency is represented by the stored data.

Fig. 2 shows that there are two channels, each as described above. Prior to the FDMC, the RF signal is split into two identical signals with 90° of phase separation. This is called quadrature phase, or I and Q. The Q signal trails the I in the time domain. These signals use the two channels of the FDMC. This effectively doubles the bandwidth of the system because two bits are stored per clock

cycle, one per nanosecond at 0.5-GHz. Thus a 64K chip would provide 65.5 μs of storage capacity.

The other operating mode of the FDMC is as a conventional static RAM. The memory contents can be accessed directly from off-chip for analysis and other ECM. The memory can be directly loaded from an off-chip CPU to effect complex signal synthesis. Conversion from serial mode to RAM mode is accomplished by switching a single pin, which turns the RAM controls over to external pins. This is termed the direct-access, or RAM, mode. In serial mode, the RAM controls are generated on-chip by the control shift registers (CSR's). To reduce pin count in RAM mode the memory is organized into 16-bit words and the data pins are bidirectional.

The RAM macrocell was designed in a separate effort as a member of a standard cell library. It was designed with a

16×128 organization, with a Y select to enable either the right or the left side of the macrocell. The programmability of the organization was effected simply by bringing out both left and right Y-select signals to a circuit which would activate both sides in serial mode for 32-bit words, and respond to an external Y-select signal in RAM mode for 16-bit words.

The row or X select was by a conventional NAND gate decoder. The bit cell was of a full six-transistor CMOS design. The bit lines were precharged. Each of the 32 columns had a sense amp with current-mirror style load, a data latch, and a tristate output buffer.

The cycle time of the RAM was required to be under 28 ns. Approximately 56 ns were available for a WRITE operation, address change, and READ operation. This occurs during the 64-ns freeze period of an SSR.

Other components of the FDMC shown in Fig. 2 include: the delay shift registers (DSR's), which can be used to add increments of eight clock cycles or 16 ns of storage time; CSR's, which control the freeze/shift operation of the SSR's and other critically timed operations; and the phase detection and correction circuitry, which among other things can be used to tie smoothly the end of a stored signal to the beginning.

III. DYNAMIC CMOS SHIFT REGISTERS

Simple dynamic shift registers permit maximum clocking frequency. The nodal leakage currents (dominated by junctions) and capacitance (dominated by the transistor gates and junctions) are small enough (about 1 nA and 0.1 pF, respectively) to result in a time constant for voltage degradation of around 1 ms. Thus the use of slow static shift registers was avoided. In actual practice, the time needed for data retention is a few nanoseconds.

Another interesting issue in the use of these shift registers is clock feedthrough. This can slow the operation of the shift register and, in the final stages of the chip, introduce an unwanted frequency component into the spectrum of the output signal. In a CMOS circuit form, both clock and its inverse are present, and their contributions to the data voltage level can largely cancel. The use of BF, for source–drain doping of the p-channel transistors allows the overlaps of p- and n-channel gates to be nearly equal. With the use of oxide spacers, the clock feedthrough could be even more closely balanced.

IV. CLOCKING

Clocking the shift registers near their maximum frequency, determined by the time required for inverter/XMG propagation delay t_D, eliminated the need for two-phase nonoverlapping clocks, and thus on the prototype chip single-phase clocking was used. Consecutive shift register stages are simultaneously turned on and off, respectively. The delay t_D prevents a second inverter from switching before its preceding XMG is off. This clocking method
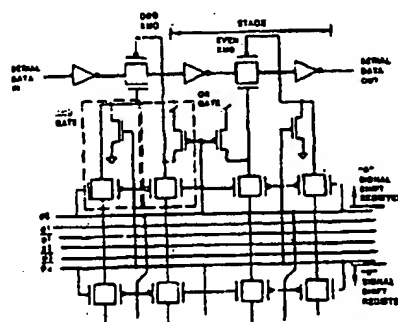
Fig. 4. Alternate clock gating scheme.
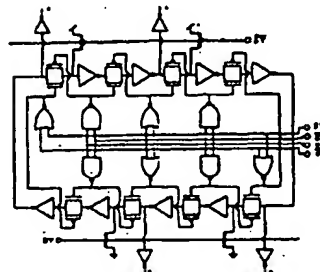
Fig. 5. A complete four-stage CSR to illustrate operation. Actual CSR's were 16, 32, and 64 stage.

requires that clock switching times and misalignment of clock and clock inverse be less than 0.5 ns. Otherwise, all transmission gates will be on simultaneously long enough to permit data to slip forward in the register. Theory and experiment both suggest that data will always slip in a long enough continuously shifting register. But, as edge speed and misalignment become significantly less than t_D, the number of stages required for slippage becomes much larger than 32.

The successor chip used a two-phase clock line layout. This was done solely to permit easy functional testing with slow edge speeds. For this testing, a two-phase nonoverlapping clock will be used, while at application frequency the lines will be tied in pairs and a single-phase clock used.

The prototype chip had 444 shift register stages each with about 0.1-pF capacitance for clock and clock inverse. This relatively large capacitance had to be driven at 500 MHz with a 5-V swing. A 0.5-ns 5-V swing required a 0.5-A pulse of clock current in each line every 1 ns. Most of the 1 W of power dissipation on the chip was in the

Fig. 6. Probed output of (a) $\overline{CLKEN}$ and (b) SY.

Fig. 7. Timing diagram of SSR operation. 1) SY pulse straddles 0th (or 64th) clock pulse. 2) RESA pulse causes SSR data to be transferred in a 32-bit word to the A RAM. 3) WESA pulse causes RAM data from A or b to be transferred into the SSRA. 4) Store addresses (six for the 1K RAM chip) must be stable. 5) Recall addresses stable.

clock enabling circuits. To avoid signal attenuation and electromigration in the clock lines, 75-μm-wide second metal was used.

This situation has been significantly improved in the successor chip, with total chip clock line capacitance reduced from about 120 pF in two clock lines to 50 pF divided among four clock lines. This was accomplished by taking full advantage of a more advanced process and by using an alternate clock gating circuit, shown in Fig. 4. Compare this transmission gate logic circuit to the conventional NAND/NOR circuit shown in Fig. 3. Merging of source–drain regions is one reason why capacitance could be reduced.

V. CONTROL SHIFT REGISTERS

Clocks to alternately shifting SSR's must be enabled at times within 0.5 ns. This precision was obtained with looped SSR-like shift registers. Fig. 5 shows a four-stage

CSR for illustration. Actual CRS's were 16, 32, and 64 stages. The external SY strobe acts to freeze and initialize the CSR's during one clock cycle. One-half of the CSR is loaded with ZEROES, the other with ONES. Then the data circulate producing a distinct square wave at each CSR stage output. After 64 clock cycles, the CSR data should return to the original positions. At 63 clock cycles, another SY pulse, again one clock cycle long, ensures this. Thus, if any one of a group of CSR's becomes unsynchronized, this will last less than 64 clock cycles. A common SY pulse will synchronize all internal functions of many chips. Identical tapered drivers, with delay of one clock cycle, were placed at opposite points on the 64-stage CSR to generate the clock enable signal and its inverse, without offset. Fig. 6 shows SY and the clock enable inverse signal from the CSR64.

In addition to square waves, precisely timed pulses, e.g., RES (refer to Fig. 3), may be generated by taking any pair of taps from one CSR as NAND inputs. Both the rising and

Fig. 8. Phase correction circuitry.

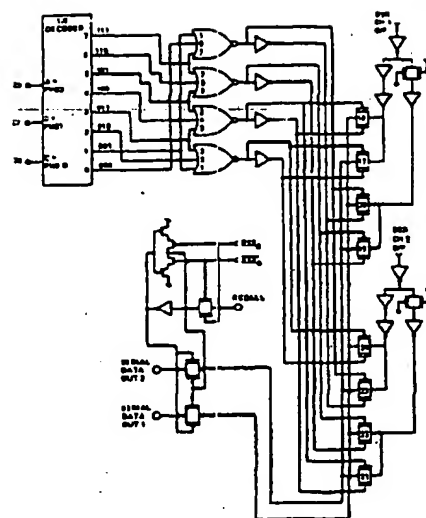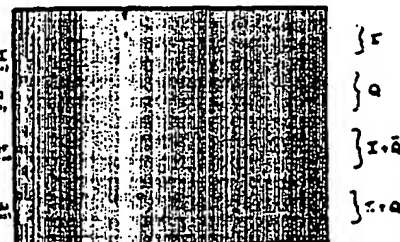Fig. 9. Output with 45° of phase correction, showing the half levels achieved by the circuit in Fig. 7.

the falling edges of the pulses exhibit sub-0.5-ns precision. Off-chip controls are sampled and latched at proper timing in this way. The CSR's control the RAM's during operation in the serial mode, but, during RAM mode, external control and data-bus drivers allow 16-bit direct access to stored data by an external computer, as discussed earlier. Fig. 7 shows a timing diagram of chip operation in serial mode.

VI. PHASE CORRECTION

"Head-to-tail playback" of a stored RF pulse to create a pseudo-CW output is a common DRFM task. This operation mode of a DRFM system is called recirculation. Phase errors occur at the start/end boundary, and to reduce these a phase correction is needed at each boundary. As described above, external circuits drive the two chip RF inputs in phase quadrature ($I-Q$). Phase detection circuits, in both channels, latch samples to measure the phase difference between the start and end of a stored pulse. Phase-correction circuitry, using these data, acts to swap and add channels, with or without inversion, at the RF outputs, thus changing the phase in 45° steps. Fig. 8 shows this circuitry. Chip output at 45° phase correction shows the half-level voltage that was achieved (see Fig. 9). This will produce a purer frequency spectrum in the output as well as achieving the fine phase resolution. Together, these circuits reduce the phase discontinuities inherent in recirculation.

VII. DELAY SHIFT REGISTERS

As a major function of a DRFM system, the chip acts as a delay element with delay as long as desired. The SSR's give a minimum delay of 64 clock cycles with a resolution of 32 cycles through RAM address manipulation. This manipulation involves the storing of data in one RAM block, followed by the recall of data from the opposite block, all within one SSR freeze period. The 56-stage DSR's, with taps every eight stages, provide a minimum delay of only a few t_D with a resolution of 16 ns.

VIII. OUTPUT DRIVE

Output signal attenuation was used to effect full bandwidth transfer. This was necessary because small-geometry transistors were used in the SSR's to minimize clock load. These were not capable of driving external loads to full CMOS logic levels, nor was this necessary, in the DRFM system. An equivalent output source resistance on the "RF data output" of about 500 Ω is obtained with a transmission-gate output stage. This is coupled to a 25-Ω transmission-line load. Thus voltage division is used to maintain frequency at the output. Signal amplitude is restored by an external linear amplifier. By using tapered drivers, carefully laid out, significant drive at 0.5 GHz could be achieved, at the cost of some latency. Since this was not needed in this application, such drivers have not been explored in detail.

IX. SIMULATION

For circuit simulation, there was concern that the available analog tool SPICE, with the MOS2 and MOS3 models, would not be useful due to the high speed of the circuits and the fundamental quasi-static nature of the models. However, the short channels kept the quasi-static approximation valid, with Ward's criterion [3] met. Simulation problems with the use of SPICE2G.5 MOS2 and MOS3 models were chiefly related to modeling internodal capacitance and matching dc $I-V$ curves in all regions of terminal bias. Fig. 10 shows the results of simulation of the
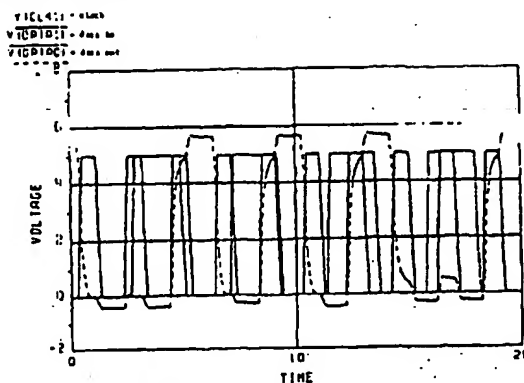
Fig. 10. SPICE MOS3 simulation of SSR operation.

SSR circuit, predicting operation above 0.5 GHz. To aid in hand calculation, an empirical expression for the delay through simple stages was derived and used. Thus these circuits did not present unusual simulation problems.

X. TESTING

Wafer- and package-level testing of IC's generally explores one or more of four aspects of the chips: functional correctness; performance characteristics, including speed and power dissipation; yield; and reliability. At this writing, the first two areas have been addressed. The first, functionality, could be investigated at the wafer level only on the low-speed portion of the chip because of limitations of existing wafer probe apparatus. Only low-frequency testing could be performed and the shift registers work only with signals as described above. However, most of the chip subcircuits could be checked.

By packaging chips which passed all wafer probe testing, the high-speed circuits were tested. This was performed at reduced frequency but with edge speeds approaching those in the eventual application. In fact, edge alignment of clock signals is as important as edge speed. Edge speeds of 1 ns and alignments of under 0.5 ns were achieved. All shift registers have been functionally verified in this way. Signals have been stored in the memory, using the shift registers, and later recalled from memory through the shift registers. This testing included wafer probing of internally generated signals such as "clock enable," shown in Fig. 6. Existing probes both affect circuit operation and distort the signal being observed, due to capacitive loading and independence mismatch, respectively.

Finally, testing at the application frequency, 0.5 GHz, has been performed with packaged devices. Full chip operation has not yet been tested, but proper operation of the shift registers has been confirmed at 0.5 GHz.
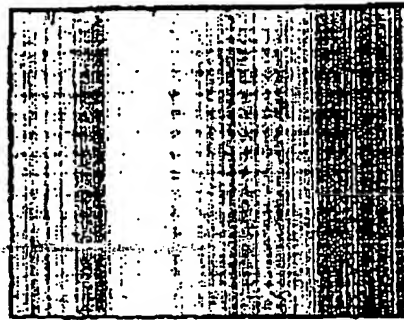

Fig. 11. Test fixture custom made for high-speed tests.

The performance of this testing has proven to be one of the more involved and expensive aspects of the project. Complex test fixtures have been built from scratch to permit the generation of the clocks and other signals in a usable form. Fig. 11 shows one of these fixtures. ECL and GaAs components have been used, though these have met the needs narrowly or not at all. A CMOS clock driver chip has been planned to solve some of the problems of both test and application.

XI. CONCLUSION

This work heralds the entry of CMOS VLSI chips into the field of RF systems. With f, in the range of 10 GHz, the performance of 1.25-μm CMOS can compete with or surpass all other production technologies. A functionally complex 0.5-GHz CMOS chip has been built and the high

speed demonstrated. An ECL-based system will be replaced by a CMOS-based one featuring twice the operating frequency, one-tenth the cost, one-twentieth the part count, and one one-hundredth the power dissipation. Single-input sample rates of over 1 GHz appear possible with 1.25-μm technology and many applications in radar and related EW systems are to be expected in the near future. Issues, not normally of concern to the CMOS chip designer, such as on-chip interconnect inductance, can become important. Test problems new to the CMOS digital test engineer are significant and major efforts are needed to develop wafer testing at high frequency and to perform efficient testing of packaged parts.

REFERENCES

[1] O. Lowenschuss, "Coherent digital RF memory—A new signal processing component," in *IEEE Proc. Nat. Aerospace Electronics Conf. (NAECON)*, vol. 3, 1980, pp. 1118-1194.
[2] B. Hoffman and D. Apte, "Gallium arsenide enhances digital signal processing in electronic warfare," *Defense Electron.*, vol. 17, no. 7, July 1975.
[3] D. E. Ward, "Charge-based modeling of capacitance in MOS transistors," Stanford Univ., Stanford, CA, Tech. Rep. G201-11, June 1981.

Edward T. Lewis was born in Cambridge, MA, on November 8, 1937. He received the B.S. degree from Tufts University, Medford, MA, and the M.S. degree from New York University, New York, NY, both in electrical engineering, in 1959 and 1961, respectively. He also received the M.B.A. degree from Northeastern University, Boston, MA, in 1972.

Currently, he is Manager of Advanced Development at Raytheon's Microelectronics Center, Andover, MA. In this capacity he is providing technical direction for advanced VLSI design, process development, and CAD. Prior to this, he was Program Manager of Raytheon's AMRAAM Manufacturing and Technology Modernization programs. He has also served as a Staff Engineer in the Microelectronics Center involved in the design and application of various exploratory LSI and microwave solid-state device technologies. From 1976 to 1977 he was Manager of the Pilot Line Operations at Sperry Research Center, where his prime responsibility centered on the development of MNOS RAM's and CMOS/SOS technology. From 1972 to 1976 he managed a department at Raytheon's Missile Systems Division involved in the evaluation of materials and devices used in advanced weapons systems. During the period from 1967 to 1972 he was a Staff Member at Sperry Research Center involved in studies related to MNOS and CCD device physics. Earlier positions consisted of an Instructor in electrical engineering at NYU and Carnegie Institute of Technology and a Senior Engineer at Raytheon. He is currently an Adjunct Professor in electrical engineering at Tufts University, teaching courses in advanced VLSI design.

Mr. Lewis is a member of Eta Kappa Nu.

William M. Schnaitter received the B.S. degree in engineering physics from Cornell University, Ithaca, NY, in 1973, and the S.M. degree in electrical engineering from Massachusetts Institute of Technology, Cambridge, in 1985.

He has been with Raytheon Company since 1973 and is currently with the Advanced Development Department at the Corporate Microelectronics Center in Andover, MA. His work is primarily in design of custom high-performance CMOS integrated circuits.

Bruce E. Gordon received the B.S.E.E. degree from the University of California, Berkeley, in 1956.

He served in the National Guard from 1948 to 1957 performing various radar and communications assignments. From 1952 to 1954 he was employed as a Broadcast Engineer at Radio Station KMYC, Marysville, CA. From 1955 to 1956 he was an Associate Engineer at Land-Air, Inc., Cincinnati, OH, working on FM telemetering receivers. He has been with Raytheon Company, Goleta, CA, since 1956, where he was an Engineer from 1956 to 1963, a Senior Engineer from 1964 to 1974, and is currently Principal Engineer in the Advanced Technology Department. In recent years he has been involved in implementing the design and construction of a series of compact digital RF memories combining RF and subnanosecond digital circuitry. These memories have become a Raytheon product line and many are flying in advanced ECM equipment. He has five patents issued in DF receiver and digital RF memory technology. He is the author of major portions of the Civil Air Patrol national search and rescue manual and a recognized authority on electronic search for rescue.

ADVANCED SELF-TIMED SRAM PARES ACCESS TIME TO 5 NS

WITH OVER
2 KBYTES PLUS
PARITY
CHECKING, A
BiCMOS ECL-
COMPATIBLE IC
EASES FAST
CACHE DESIGNS.

DAVE BURSKY

The performance of high-speed computer systems often depends on how fast data can be retrieved from memory. For super-minicomputers, mainframes, and specialized computing systems, fast-accessing static RAM (SRAM) cache memories, as well as other high-speed memory subsystems, are still a key requirement to achieve system cycle times of less than 10 ns.

However, as memory chips push the process limits to achieve the sub-10-ns access times, system signal skews and other timing problems often rule out the use of standard asynchronous SRAM designs. Consequently, a new memory type dubbed the self-timed SRAM was born—a structure well suited for synchronous system design. Although they're not the first self-timed SRAMs to be released, a family of five 2-kword-by-9-bit biCMOS chips with ECL 100K-compatible input/output lines from National Semiconductor offers the highest density with more features and faster access
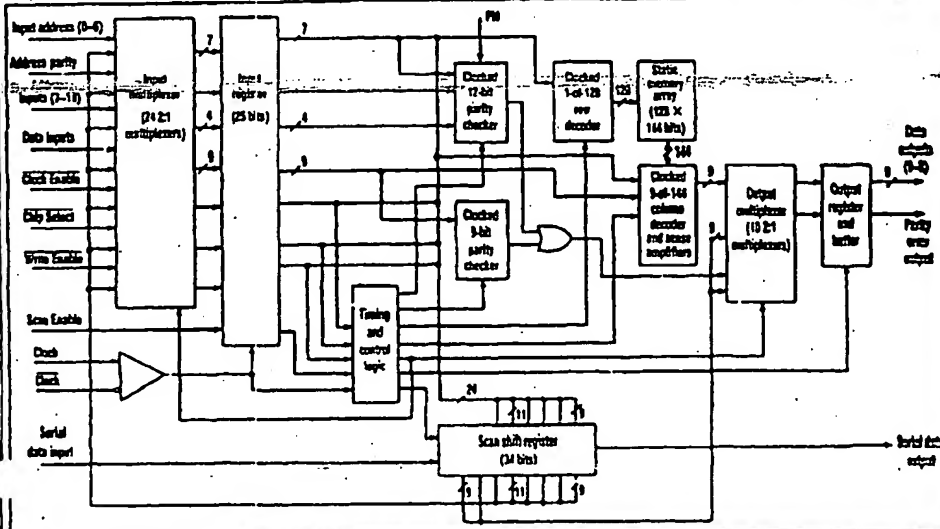
1. THE SELF-TIMED SRAM includes a static memory array, input and output registers, parity-checking logic for both data and address inputs, and serial scan shift logic. All circuits are controlled by a clock input and an on-chip timing generator.

SELF-TIMED
STATIC RAM

times than potential competitors.
Offering access times as fast as 5 ns, the NM4492, which has 100K ECL I/O lines but operates from a -5.2-V supply rail, also comes in 7- and 10-ns versions. Two other versions of the RAM, which operate from the standard -4.5-V 100K ECL power supply, come in 7- and 10-ns versions. Part of the fast access time can be attributed to the mixed bipolar-CMOS design that employs 0.7-μm minimum features and supplies ECL I/O levels. The rest stems from the novel self-timed architecture and separate data-input and data-output buses to minimize delays.

Not only can the memory chips access their data in such short amounts of time, but the systems they're used in can actually cycle in the same time frame. That's because the RAMs are fine tuned for synchronous system operation. The systems can therefore operate much faster than with standard SRAMs, because various setup and hold times appear much shorter than with asynchronous or other self-timed static memories.

Such chips can greatly improve the performance of register files, writeable control stores, cache and cache-tag memories, and address-

translation lookaside buffers. Two early adopters of the advanced self-timed memories—Control Data Corp. and Convex Computer Corp.— have embedded the chips in advanced computer systems. They project the same performance couldn't have been achieved with any other commercial memory chip.

With the advanced self-timed RAMs, ECL processors can achieve a memory system speedup of 60 to 150% vs. the use of standard SRAMs, and close to a 150% speed improvement vs. the use of first-generation self-timed RAMs offered by other companies. Not only can the systems run faster, but fewer chips will be needed, which can reduce board space, lower power consumption, or allow larger memory subsystems in the same space. Even higher-capacity and more feature-laden chips will be forthcoming.

The 4492 or 100492 pack the largest amount of storage on one chip for any self-timed ECL memory—2048 words by 9 bits. And according to Charles Hochstedler, product planning manager of National's static memory division in Puyallup, Wash., the performance comes at a modest power level for the speed, thanks to

the biCMOS circuit structures and process. The 5-ns version consumes about 2.5 W (2.7 W worst-case), while the 7-ns device draws a maximum of 2 W when running at full speed. Power consumption is somewhat speed dependent, and it drops to less than 1.8 W when operating frequencies drop below 100 MHz. An idle mode with the clock stopped drops the power drain to about 650 mW.

The ninth bit of the data word is a parity bit. To ensure the integrity of incoming data, the RAM also includes on-chip parity checking logic. Unlike all other RAMs, though, the National RAMs will also check the address inputs for a parity error to ensure that the wrong location isn't being accessed. The RAM checks for odd parity on the data-input field and for either even or odd parity on the 11-bit address field (depending on the state of the Parity-Mode pin).

If either parity check determines an error is present, the chip's Parity Error flag is set. The polarity of the error-output flag simplifies the emitter-dot-ORing of several open-emitter outputs so the delay can be minimized. Address parity detection can be disabled only if data parity is desired, or both parity features can be ignored if the system doesn't implement parity.

Furthermore, the RAMs contain scan registers that support system scan diagnostics. Each chip has a separate serial-scan input and output and a 34-bit serial-scan shift register that enables users to observe the state of the input registers and force the state of the input and output registers.

With the scan path, systems can test interconnectivity and bus-conflict faults on the address, data inputs, and control lines leading to the memory chips. The systems can also test data outputs and the parity error output line from the chips.

The serial input can also be used in writeable control-store subsystems to load the memory during system boot-up, thus simplifying the circuit-board layout and eliminating a wide parallel bus for data inputs.

Self-timing is a scheme adopted by National and other companies to re-
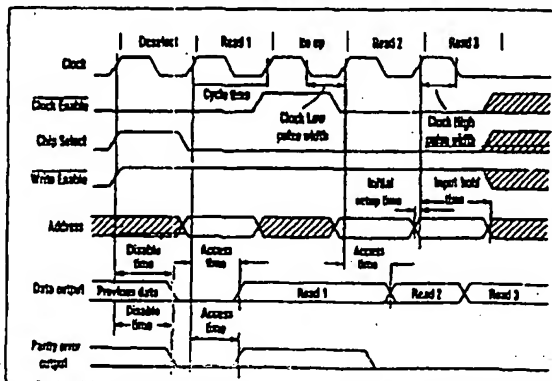
2. THE READ operation in National Semiconductor's self-timed SRAM requires that the Chip Select line be low and the Write Enable line be high before the rising edge of the Clock signal locks the data and addresses into the chip's input registers. Data will reach the chip's Q outputs a minimum of 2.5 ns and a maximum of 5 ns later. The data won't change again until another read operation occurs.

SELF-TIMED STATIC RAM

duce timing skews and minimize setup and hold times, says Hochstedler. In the approach, all input signals (address, data, and control) are latched into on-chip registers by a low-to-high transition of the clock signal *(Fig. 1)*. By latching all inputs, the setup and hold times can be minimized. In the case of the 4492, the combined setup and hold time required is as little as 2 ns; the 100492-i requires just 2.5 ns.

A typical asynchronous SRAM responds to an address change by supplying new read data one address access time after the change. Self-timed memories respond to new inputs only when clocked. The clock synchronizes the memory operation to the system timing, making possible much tighter signal timing windows. The asynchronous memories are applied mostly to systems in which a timing signal isn't easily derived from available control signals.

One major advantage of self-timed memories is the control of the write cycle by self-timing circuits on the memory chip itself. This control eliminates concerns that the write-pulse width might be too short to complete the data write. In a slow system, that may not be a critical concern. But in high-speed systems with clocks that may range from 50 to 200 MHz, even a few nanoseconds of skew could degrade system performance. By including input registers on the chips, the system buses are free to start the next bus cycle even before the memory finishes its access once data is locked into the chip.

In most self-timed RAMs, the output register is clocked by the same timing signal that controls the input register. That causes the RAM to appear in the system only as one pipeline stage. Although that might suit some applications, there are many instances where the system timing demands a different approach. For that reason, National Semiconductor self-timed RAMs use a separate self-timing circuit that triggers during the write pulse and then delivers a delayed timing signal to the output register.

With the delayed signal, the registers can hold the output data valid for an extended portion of the read cycle *(Fig. 2)*. The extension of the time that data stays valid eases the system read-timing requirements.

Another timing improvement can be derived from interleaving read and write operations in a mode called hidden write. By keeping the output register active (with the last read data output) during a write cycle, the RAM greatly simplifies the timing of interleaved memory architectures.

In fact, if the machine's cycle time is at least twice the access time of the memory chip, both address read and write cycles can be squeezed into one machine cycle. Such a mode can be very useful in cache and register-file applications, where multiple sources and/or destinations may be interleaved within each machine cycle. The hidden-write mode is controlled by the Write Enable and Chip Select lines, which have slightly different characteristics than those found in an asynchronous memory.

For synchronous systems, the advanced self-timed memories also include a Clock Enable input, which simplifies the starting and stopping of pipeline operations. It reduces, and could eliminate, the requirement to gate the clock signal external to the RAM. The feature doesn't affect systems that don't need it because an on-chip pull-down element will ensure normal operation if the clock enable is unused.

PRICE AND AVAILABILITY

The advanced self-timed SRAMs are immediately available in production quantities. They sell for $149, $105, and $86 for the 5-, 7-, and 10-ns versions, respectively, all in lots of 1000. All of the chips will initially be housed in 68-lead ceramic flat packages.

National Semiconductor Corp., 1111 West Ave. SE, Puyallup, WA 98373, Charles Hochstedler, (206)357-4797. CIRCLE 503

HOW VALUABLE?
HIGHLY ......
MODERATELY ......
SLIGHTLY ......

CIRCLE
...
...
...

A Video Codec LSI for High-Definition TV Systems with One-Transistor DRAM Line Memories

TOMOJI TAKADA, TAKESHI OTO, KAZUKUNI KITAGAKI, NAOYUKI HATANAKA,
TATSUHIKO DEMURA, HIROMICHI FUJI, TOSHINORI ODAKA,
HIROSHI SUE, AND TADAHIRO OKU

Abstract —A video codec LSI for high-definition television (HDTV) systems has been developed. By using a time-compressed integration (TCI) encoding technique, it converts a 20.0-MHz bandwidth luminance signal and two 5.0-MHz chrominance signals into a compressed image signal at 43.6-MHz sampling frequency. It is useful in many HDTV application systems, such as a 600-Mbit/s digital transmission system, a video disk player system, or an analog transmission system. Over 260 000 elements, including a 32-kbit one-transistor DRAM line memory specially developed for this LSI, were integrated on a 12.16 × 12.16-cm² chip. A standard cell layout method and a 1.3-μm CMOS logic LSI process were used.

I. INTRODUCTION

IN RECENT YEARS, several high-definition television (HDTV) encoding methods, such as MUSE [1], HD-MAC [2], and time-compressed integration (TCI) [3], have been proposed for use in the next generation of TV systems. The TCI method is one of the most simple and flexible. 15-bit format, a more complicated image processing method, like a discrete cosine transformation or a vector quantization, can be applied. A codec LSI which uses the TCI encoding method has been developed as the first step of the HDTV encoding systems. The major problem in the realization of the codec LSI was the high density and large area of line memory circuits needed. Therefore, a one-transistor/cell DRAM line memory was developed. This paper describes a codec LSI [4] which uses the TCI encoding method, with an emphasis on the development and characteristic usage of one-transistor DRAM line memories.

In the TCI method, a luminance signal Y and two chrominance signals, Cw and Cn, are multiplexed to form a TCI format signal, which can be transmitted entirely within the bandwidth of Y.

TABLE I
BANDWIDTH AND SAMPLING FREQUENCY OF HDTV

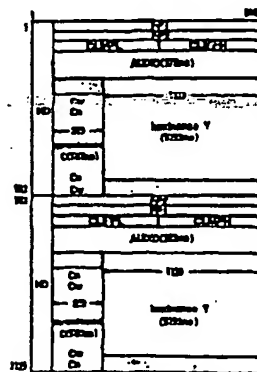|		Bandwidth	Sampling Frequency
LUMINANCE	Y	20.0 MHz	44.0 MHz
CHROMINANCE	Cw	5.0 MHz	12.15 MHz
CHROMINANCE	Cn	5.0 MHz	12.15 MHz
TCI Signal		20.0 MHz	43.5 MHz

Fig. 1. TCI format.

II. HDTV SIGNAL SPECIFICATION AND TCI FORMAT

In the TCI encoding method, the two chrominance signals Cw and Cn are compressed by a factor of 4 along the time axis, and their sampling frequencies are raised to 43.6 MHz. These signals are then multiplexed into the horizontal blanking period of the luminance signal Y in
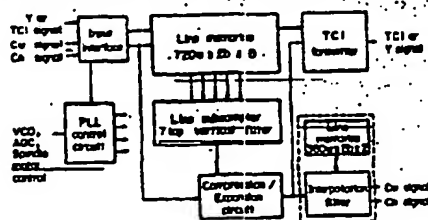
Fig. 2. Block diagram.

Fig. 3. Peripheral interface to codec LSI.

line-sequence form. The bandwidth and the sampling rate of the resulting TCI signal are identical to those of the luminance signal Y, 20 and 48.6 MHz, respectively. Thus, the entire HDTV signal can be transmitted within the bandwidth of the Y signal. The codec LSI can convert the Y, Cn, and Cw signals into one signal in the TCI format, and vice versa. Table I shows the bandwidth and sampling frequencies of HDTV and TCI signals used in this codec LSI.

Each frame of a time-compressed HDTV signal is composed of two fields or 1125 lines, and each line is 1440 pixels wide. Each line has three subfields: the Y signal subfield (1120 pixels), the Cw/Cn subfield (280 pixels), and the horizontal synchronization or HD subfield (40 pixels). Frame synchronization signals ($FP1$ and $FP2$), reference clamp level signals ($CLMPL$ and $CLMPH$), as well as audio signals are also inserted at the beginning of each field. Fig. 1 shows a TCI format used in this codec LSI.

demultiplexed into two 24.3-MHz signals before being applied to the input interface, as shown in Fig. 3.

B. Vertical Filter

In the TCI format, the Cw and Cn signals must alternately share the Cw/Cn subfield of the compressed HDTV frame. Therefore, the codec LSI must perform vertical subsampling on the chrominance signals in the encoder mode. The vertical filter is used as a prefilter for vertical subsampling in the encoder operation, and cuts the vertical spatial bandwidth in half to prevent aliasing effects. The transfer function of the vertical filter F_v was determined empirically from a careful evaluation of the decoded image obtained from a breadboard prototype:

$$F_v = 1/2 Z^0 + 9/32(Z^{-1} + Z^1) - 1/32(Z^{-3} + Z^3) \quad (1)$$

where Z is unit line delay. The multiplication in (1) is realized by shift registers and adders. Six 720-word × 8-bit line memories are used to obtain the seven taps.

III. ARCHITECTURE

This codec LSI can convert the Y, Cw, and Cn signals into one signal in the TCI format and vice versa. In the encoder mode, it has three functions: the filtering of the C signals, the time compression of the filtered C signals, and the multiplexing of the Y, C, and additional synchronizing signals. In the decoder mode, it also has three functions: the time expansion of the derived C signals, the interpolation of the missing C signals, and the detection of the synchronization signals.

For these functions, the codec LSI consists of seven functional blocks: an input interface, line memories, a vertical filter, a time-compression/expansion circuit, a TCI formatter, an interpolation filter, and a PLL control circuit. The functional blocks of the LSI are illustrated in Fig. 2 and are described below.

A. Input Interface

The input interface block receives input signals and distributes them to the other functional blocks. The Cw and Cn signals are applied directly to the input interface. The Y signal, however, is too fast to be treated easily with TTL peripheral circuitry. Therefore, the Y signal must be
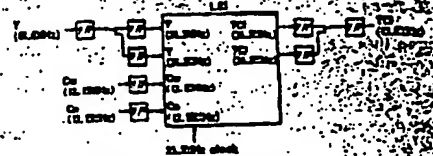
C. Time-Compression/Expansion Block

In the encoder mode, two 12.15-MHz signals, Cw and Cn, are compressed by a factor of 4 along the time axis and their sampling frequencies are raised to 48.6 MHz. In the decoder mode, they are expanded along the time axis and their sampling frequencies are restored to their original 12.15 MHz. The time-compression/expansion block implements this time axis transformation. It consists of a shift register and clock frequency switching logic. Because the video signal in raster scan format is an uninterrupted signal, it was possible to implement this shift register as a dynamic circuit.

D. Interpolation Filter

As mentioned previously, every other Cw and Cn sample is lost during subsampling in the encoder operation. The interpolation filter accomplishes the vertical linear interpolation on adjacent lines to recover the missing C samples in the decoder operation, and outputs the restored Cw and Cn signals. The transfer function F_i is

$$F_i = 1/2(Z^1 + Z^{-1}). \quad (2)$$

For these operations, this block has two 360-word × 8-bit line memories and an adder.

E. TCI Formatter

In the TCI formatter block, the compressed C signals are multiplexed into the horizontal blanking period. Audio signals, vertical synchronization signals (frame pulses *FP1* and *FP2*), a horizontal synchronization signal (*HD*), and reference clamp level signals are also multiplexed as part of the TCI format.

F. PLL Control Block

The PLL control circuit detects the horizontal synchronization signal *HD* and frame pulses *FP1* and *FP2* in the input TCI signal during decoder operation. It compares the phase of the input TCI signal with that of the system clock and outputs a digital control signal for an external VCO circuit which generates the system clock. Furthermore, it can control an automatic level control (ALC) circuit and an automatic gain control (AGC) circuit in an analog transmission system, and can control a spindle motor in a video disk system. Because of the integration of this PLL control circuit, peripheral circuits are drastically reduced in the TCI decoder system. This block takes up roughly one half the area of the logic portion of the codec LSI.

IV. LINE MEMORY

Line memories, as used in video signal processors, exhibit the following characteristics:

1) high-speed operation,
2) relatively large memory capacity (for a logic LSI),
3) cyclic, uninterrupted read/write operation (rewritten every horizontal scanning period),
4) FIFO access only (no random access needed), and
5) fabrication process and circuit parameters compatible with logic circuits (especially with standard cell circuits).

Several kinds of memory circuits including shift register [5], three-transistor/cell DRAM [6], [7], four-transistor/cell DRAM [8], and one-transistor/cell DRAM [9] have been used to implement line memories. Because several tens of thousands of memory cells are required to implement the line memory in this codec LSI, a one-transistor/cell DRAM circuit is the most profitable by virtue of its high density. Because of characteristic 3) mentioned above, the automatic refresh cycle is executed every 0.03 ms (= 1/30/1125 s) without any refresh control circuits.

Fig. 4 shows a block diagram of the line memory implemented in the codec LSI.

Because line memory access is completely sequential, 1:8 serial-to-parallel converters and 8:1 parallel-to-serial converters are used at the input and output sides of the memory cell array. This design has two remarkable benefits:
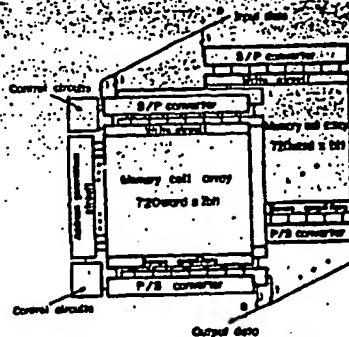
1) the internal read/write operation cycle for the
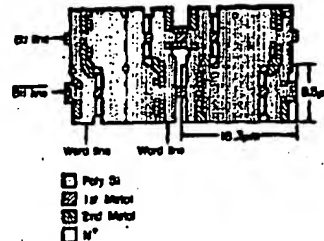

Fig. 4. Block diagram of the line memory.

☐ Poly Si
▨ 1st Metal
▧ 2nd Metal
☐ N+

Fig. 5. Memory cell pattern.

Fig. 6. Minimum cycle time.
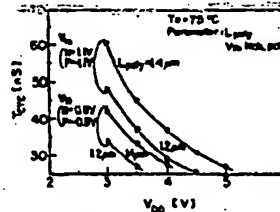
memory cell array is relaxed, and
2) by taking advantage of the eight clock cycles in one memory access cycle, only level-sensitive synchronous circuits are needed to generate the internal memory control signals.

Because only level-sensitive circuits are needed, testability is enhanced and process sensitivity is minimized.

In order to make the circuit parameters V_{th} (threshold voltage) and V_{DD} (supply voltage) the same as for the logic portions of the LSI, the line memories use neither word-line bootstrap circuits nor backgate bias circuits.

Fig. 7. Photomicrograph of the LSI.

TABLE II
MAIN FEATURES OF TCI CODEC LSI

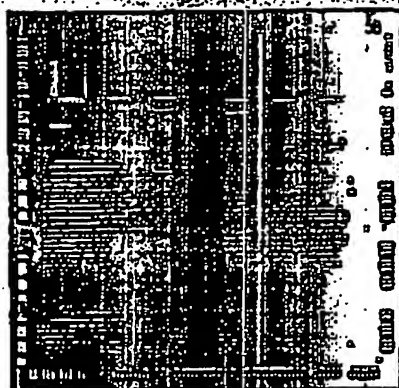|	
Technology	1.2 micron P-well CMOS
Chip size	12.16mm x 12.10mm
Number of elements	Total 288K elements
— Random	6,000 Standard Cells
— Line Memory	52K bit
Clock frequency	24.3 MHz
Power Consumption	1.0W (typ.)
I/O Interface	TTL Compatible

Fig. 8. Single board TCI decoder.

Furthermore, in order to be able to use the same fabrication process, neither the thin oxide layer for cell capacitance nor the high-capacitance junction [7] is used. Instead, cell capacitance is realized with polysilicon gate capacitance and a $^+$-p-well junction capacitance. Fig. 5 shows the memory cell pattern. An 83.5-fF cell capacitance is realized in the 8.5×16.3-μm^2 cell area.

Fig. 6 shows the measured minimum cycle time as a function of supply voltage V_{DD}. The memory circuit operates steadily at 3 V, in spite of the constraints in fabrication process and circuit design mentioned above.

V. LSI PERFORMANCE

A mixed hierarchical layout approach was used in designing the codec LSI. The line memory and shift registers of the time-compression/expansion block were laid out without the aid of automatic layout tools, and the random logic portions were designed by a hierarchical standard cell approach. Fig. 7 shows a photomicrograph of the LSI.

A 1.2-μm p-well CMOS with double-level-metal interconnection technology was used to integrate 288K elements, including a 52-kbit line memory, on a 12.16×12.10-mm^2 chip. The chip was mounted on a 209-pin PGA package. At 24.3 MHz the measured power consumption was 1.0 W. The features of this LSI are summarized in Table II.

VI. APPLICATION SYSTEMS

This codec LSI has three operation modes corresponding to three HDTV applications:

1) digital image transmission system,
2) analog image transmission system, and
3) video disk player system.

Characteristics for each operation mode are described below.

A. Digital Transmission Mode

The signal sampling rate of the TCI signal was chosen to be 48.6 MHz. Consequently, the overall bit rate of the 8-bit TCI signal is 388.8 Mbit/s. Thus, by using this codec LSI, an HDTV image signal can be transmitted on the widely used 400-Mbit/s digital transmission line.

In the digital transmission mode, the codec LSI accomplishes its primary functions: encoding and decoding. In both encoder and decoder systems, few peripheral circuits are needed. The digital transmission system is suitable for considerably long-distance HDTV image transmission.

B. Analog Transmission Mode

At the decoder site of the HDTV analog transmission system, system clock regeneration and analog level control are needed. The on-chip PLL control circuit is used for this purpose. It controls the frequency and phase of the system clock, and controls the gain and dc levels for the peripheral analog circuits in the decoder system. Because of the on-chip system control circuits, the peripheral circuitry for the TCI decoder system is significantly reduced. For example, the decoder circuit for an HDTV analog transmission system can be realized on a single board using the codec LSI (Fig. 8) whereas previously, using only

James D. Trotter (S'58-M'63-S'70-M'70) was born in Sweetwater, TX, on August 8, 1933. He received the B.S. degree from Mississippi State University, State College, the M.S. degree from Stanford University, Stanford, CA, and the Ph.D. degree from the University of Texas, Austin, all in electrical engineering, in 1960, 1962, and 1970, respectively.

In 1962 he joined the Research and Development Laboratory of Fairchild Semiconductor Corporation, Palo Alto, CA. As an Engineer in the Digital Integrated Circuit Development Group, he participated in the development of the epitaxial micrologic process and was personally responsible for the design of the first circuits of the Fairchild DTL 930 series. He joined General Micro-Electronics (GME) in 1963

as Head of the Digital Integrated Circuit Group. He designed the original DTL and TTL circuits of the GME series and supervised the development of other RTL and ECL circuits. After GME's entry into the development and production of integrated MOSFET circuits, he was responsible for the coordination and development of the company's MOSFET design principles. He shared in the award of two patents in MOSFET circuit design. In 1965 he joined North American Rockwell as an Assistant to the Director of the Micro-Electronics Development Laboratory, and helped coordinate the development and implementation of MOSFET design and fabrication techniques. After receiving the M.S. degree he rejoined North American Rockwell and led their silicon gate process development. In 1971 he joined American Microsystems, Inc., Santa Clara, CA, as Director of Research and Development. He is presently the Vice President of Technology Development at American Microsystems, Inc.

MNOS-BORAM Memory Characteristics

ROBERT J. LODI, H. A. RICHARD WEGENER, SENIOR MEMBER, IEEE, MILLICENT B. BOROVICKA,
BERNARD B. KOSICKI, MEMBER, IEEE, THOMAS A. POGEMILLER, MEMBER, IEEE, AND
MARSHALL W. EKLUND

Abstract — This paper describes the characteristics of a block-oriented random-access memory (BORAM) system which uses a custom-designed 24-bit MNOS memory array for information storage. Delivery of two fully functional memory systems has been a significant achievement in the development of the MNOS memory technology. The organizational concepts and performance characteristics of both the memory system and the MNOS memory array will be discussed, including speed, data transfer rate, and retention.

I. INTRODUCTION

A NEW memory system, utilizing nonvolatile semiconductor MNOS information storage, organized as a block-oriented random-access memory (BORAM) [1] has been developed. MNOS technology was selected because it provides nonvolatility in addition to the other advantages of a semiconductor memory. Nonvolatility not only provides security of stored data but also provides lower system power dissipation since it is possible to power down the unselected memory arrays. Two fully populated systems are now in operation: the first is a 295-kbit system, and the second is a 590-kbit system. Both of these systems utilize a 2048-bit MNOS memory array which was custom designed for this

application. We believe that the construction and operation of these systems constitute a major milestone in the development of MNOS memory technology.

II. SYSTEM DESCRIPTION

A block-oriented random-access memory (BORAM) has an architecture similar to a normal RAM. It is organized in words consisting of a number of parallel bits. However, when an address is presented to the memory system, not one but a predetermined number of words are output sequentially. This string of words is called a block. Superficially, a BORAM functions like a drum with a very short latency time and a very short record. BORAM's are potentially more inexpensive since a savings in address decoding circuitry is realizable. In addition, BORAM's are potentially faster. In a memory system the access time is the sum of the propagation delay between CPU and memory plus the access time of the memory itself. As main memory access times become shorter and shorter, the propagation delay between CPU and memory is an increasingly larger percentage of the system access time. A block-oriented memory reduces this problem by requiring the CPU to handle blocks of words instead of one word. Therefore, the propagation time between CPU and memory is averaged over the number of words in a block. Analogous arguments are true in describing the advantages of an LSI chip designed as a BORAM over one designed as a RAM. This is particularly applicable to MNOS memories. Their long write times are effectively reduced by a factor inversely proportional to the number of MNOS transistors written simultaneously as members of the same block.
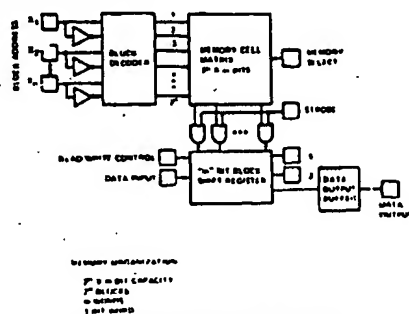
Fig. 1 shows a block diagram for a semiconductor BORAM

MEMORY ORGANIZATION

$2^n = m$ BIT CAPACITY
2^n BLOCKS
m WORDS
1 BIT WORD

Fig. 1. Generalized semiconductor BORAM.

chip organized as 2^n blocks by m words/block with one bit/word. To initiate a read cycle, the BORAM is accessed by its n-bit block address and memory select inputs. This address selects one of the 2^n blocks in the memory cell matrix, the contents are parallel transferred to the m-bit block shift register after the strobe input has been activated. The block data consisting of the same bit from each of the m words are now shifted out of the register and sensed on the data output. A write cycle is initiated by addressing the memory chip and block and activating the read/write control. Then m consecutive clock pulses are required to shift the new data into the register. When the strobe input is activated, all m words are simultaneously written into the memory matrix.

In the specific system described here, the input/output requirements were as follows:

word size	36 bits
block size	256 words
(system) access time	2 μs
read cycle time	40 μs
write cycle time	2 ms
data transfer rate	150 ns/word

Since the MNOS BORAM chip utilizes p-channel static MOS technology for the I/O shift registers, the inherent limitation of 1 MHz operating frequency of this process had to be overcome by design in order to achieve the required data transfer rate. This was done on two levels. On the MNOS memory chip, two shift registers operating simultaneously were multiplexed to result in an effective chip output rate of 1.66 MHz (600 ns/word). In addition, the chip was partitioned in such a way that each chip address accessed one quarter (64 words) of the full block (256 words). Thus, the data output from one chip is multiplexed with three others to result in a four times higher transfer rate, 6.66 MHz (150 ns/word).

A single MNOS memory chip contains 32 randomly addressable blocks 64 words wide. Its one I/O pin represents a single specific bit within a given block of words. Thirty-six of these chips are required to implement the 36-bit word. The 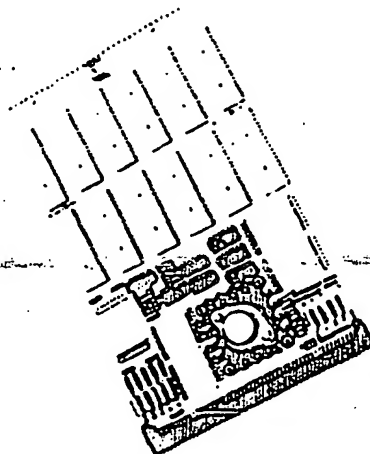minimum system module size is determined by the required number of words per block, 256, to result in a 295-kbit module formed by 32 blocks × 256 words × 36 bits.

The 144 MNOS arrays used to populate the 295-kbit single module system are packaged on 12 boards containing memory devices and hybrid interface drivers. Each board contains 12 memory arrays which are interconnected to make a 32-block × 64-word × 12-bit section of memory. Three cards are wired in parallel to produce the required 36-bit word length. Four groups of three cards are multiplexed to produce a system module.

One of the memory printed circuit cards is depicted in Fig. 2. The 295-kbit single module system is shown in Fig. 3. The 590-kbit system consists of two 295-kbit modules.
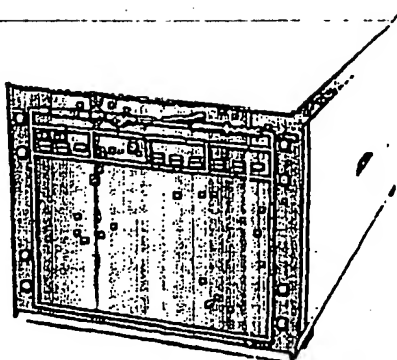

Fig. 2. BORAM system memory card.

Fig. 3. 295-k bit single module system.

III. MNOS MEMORY ARRAY CHARACTERISTICS

Based on the BORAM system requirements the MNOS memory array specifications were defined as follows:

organization	32 blocks/64 words/1 bit
data output rate	dc to 2 MHz
(chip) read access time	1.5 μs max
write cycle time	2 ms max
temperature range	-55°C to 165°C
retention time	one year minimum.

The memory chip is organized (Fig. 4) as 32 randomly addressable blocks, each having 64 serially accessed bits which represent a single bit of 64 different words. This memory array organization provided an optimum building block for the system as well as a memory chip with favorable storage density and productibility.

The MNOS transistor which is used as the memory cell is similar to typical p-channel transistors. The difference is in the nitride layer which is placed above the gate region between the gate metal and the oxide layer. Fig. 5 shows a simplified cross section of a memory transistor. The nitride layer provides the ability to change the threshold voltage of the transistor [3]. (Threshold voltage is the voltage applied to the gate to turn the device on.) This variation in threshold voltage occurs because of the injection of either positive or negative charge in the nitride layer. Negative charge trapped in the insulator reduces the magnitude of the negative voltage which must be applied to the gate to turn it on (i.e., the threshold voltage). A trapped positive charge has the opposite effect, increasing the magnitude of the threshold voltage. Data are cleared and written in the memory transistor by voltage pulses across the gate which change the threshold to either the least negative or most negative value. Application of a clear voltage, typically 25 V positive, moves negative charge from the substrate into the nitride layer and, similarly, application of a write voltage, 25 V negative, moves positive
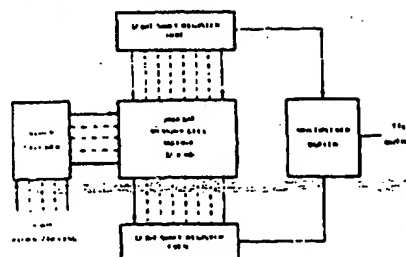

Fig. 4. Memory chip organization.
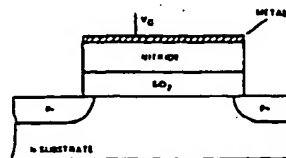
Fig. 5. MNOS transistor cross section.

charge into the nitride layer. The charge movement and charge storage are consequences of the highly nonlinear conductivity and trapping effects which occur in the memory dielectric. Once charge is trapped in the dielectric it remains until the application of a subsequent write or clear voltage alters it, thereby providing the nonvolatility characteristics of the MNOS transistor.

The MNOS memory devices used as the storage elements in the BORAM chip are arranged in a matrix configuration with the gates of a block connected in common row lines and the

Fig. 6. MNOS memory matrix.

Fig. 7. MNOS cell read/write circuit.

sources and drains of a word connected in common column lines (Fig. 6). The write operation of this MNOS memory matrix is accomplished in two distinct steps. The first consists of the application of a positive potential to the common gate electrode of a selected row of memory devices. This clear step sets all devices in this row to the same (least negative) threshold voltage. This is followed by the application of a negative potential to the common gate electrode of the selected row. The simultaneous application of data-selected-inhibit signals to the memory device source and drain lines determines which devices in this row have their threshold voltages changed to their most negative extreme. The voltage applied to the nonselected rows of devices is kept at a level such that these devices see no set potential difference across their gates. Therefore no threshold change can occur.

Reading information from the single transistor MNOS cells is accomplished by using the MNOS device as the driver transistor in an MOS inverter circuit (Fig. 7). Read voltage level (~ -8 V) is applied to the selected row of memory cells (all nonselected rows have no voltage applied). The inverter ratio is designed such that if the threshold of the memory device is at its least negative value it will turn on and the

power supply voltage (+15 V) will be dropped across the load transistor resulting in a high output level from the memory. Conversely, if the threshold of the memory device is at its most negative value the device will not conduct and the memory output will remain at its low level.

The most critical characteristic, from the viewpoint of both array and system organization, is the data transfer rate. Since the time required to read the MNOS memory cell is typically 1 μs it is necessary to read all required data for a block transfer (64 bits) in parallel into the two shift registers and then multiplex the data onto the I/O bus as required.

The 2:1 multiplexing performed on the memory array, together with the 4:1 multiplexing done at the system level, provides a method of obtaining the required data rate without overburdening any of the chip subcircuits. Fig. 8 shows the operating frequencies of the memory array and system elements that are part of the data path. The data rate of the memory array is well within the capability of conventional p-channel MOS technology.

A block diagram of the memory array organization is detailed in Fig. 9. The memory cell matrix is controlled by buffer circuits which in turn interface with the data-controlling

Fig. 8. Data transfer rate.

Fig. 9. Block diagram of memory chip.

Fig. 10. Block address decoder.
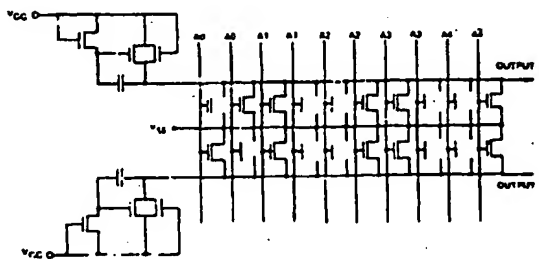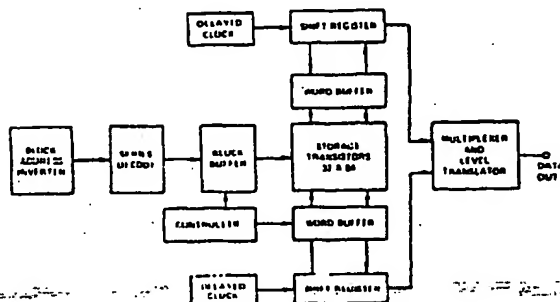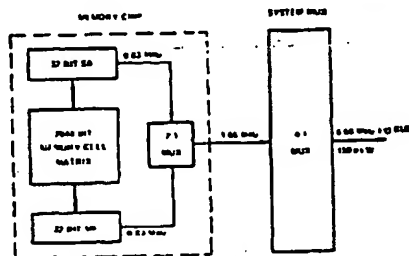
circuit elements. A discussion of the functional blocks will now be given to provide a better understanding of the circuit operation.

Five address inverters are required to accept the address inputs and provide the two requisite true and complement levels as decoder inputs. Inverter speed is critical to block

access time. A "bootstrap" technique has been used to provide a fast response, especially for the high to low output transition. The block address decoder (Fig. 10) takes the five true and five complement block address lines and selects one of the 32 rows of memory devices. The decoding is done by a selective gate pattern of five of ten parallel transis-
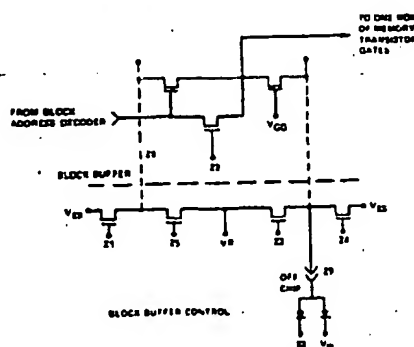
Fig. 11. Block buffer circuit.

tors. 1. he proper five address lines are energized, a negative-going bootstrapped drive occurs on the selected row address line.

All potentials applied to the variable memory transistor gate lines are controlled by the block (row) buffer circuit (Fig. 11). Inputs provided by the block decoder and controller are translated by the block buffer into appropriate potentials applied to the addressed and nonaddressed blocks (rows). The BORAM chip contains 32 block buffer circuits operated by a single controller circuit. The word buffer controls the levels applied to the source and drain lines of the memory transistors. During the READ cycle the potentials at the source and drain lines are a function of the threshold of the memory cell, but during the WRITE cycle these potentials are under control of the data stored in the shift register.

The input/output shift register used on the BORAM chip is required to perform all shift register functions (serial in/out and parallel in/out) during memory cycles. In addition, the shift register must hold data throughout the WRITE cycle. The basic shift register cell consists of two ratio-type MOS inverters and four transistors used as transfer gates. Three clocks are required to operate the shift register, two supplied from off the chip and one generated internally. A two-inverter circuit with a capacitor delay element is used to generate the required delayed clock phase for the shift register.

Data from the last stage of each 32-bit shift register are combined in the multiplexer. The multiplexer output is level shifted so the data output is a TTL level. A combinational logic controller is used to implement the required on-chip functions from the chip input control lines.

The BORAM chip has three operating modes: 1) DISABLE; 2) READ; 3) WRITE. The DISABLE mode inhibits all chip functions when a logic 0 level is applied to the chip select input. To perform a READ operation, a block address (5 bits) is applied to the address inputs. This address is decoded and selects 1 of 32 rows of memory cells. Each row contains 64 memory transistors because of the 32 row by 64 column organization of the memory array. Data from the even bits

(bit 0, bit 2, bit 4, etc.) are loaded into one 32-bit shift register, while bit 1, bit 3, bit 5, etc., are loaded into the other 32-bit shift register. These registers are each clocked at about 0.83 MHz and the outputs are multiplexed together, giving a 1.66-MHz data transfer rate out of the chip. New data are available every 600 ns on an even/odd basis from the chip. Several parameters describing the READ operation should be defined.

Read cycle—total duration from block address valid until all 64 words have been shifted out (40 μs).

Read access—total elapsed time from block address valid until the first word is available at the output (1.5 μs).

It should be noted that while the time to completely transfer data from the memory matrix to the outsideworld (40 μs) is defined as read cycle, the actual read performed on the memory transistors is approximately 1 μs.

The total WRITE cycle of the BORAM array consists of a 1-ms CLEAR (concurrent with serial loading of data into the shift registers) and a 1-ms WRITE negative/inhibit. The CLEAR sets the selected row of memory cells to their least negative threshold level. The WRITE negative/inhibit sets the selected cells to their more negative threshold value or inhibits that change depending on the input data held in the shift registers.

A photograph of the 198-X 186-mil memory chip is shown in Fig. 12. The functional circuit blocks that have been discussed are designated on the photograph. All controlling inputs, with the exception of two required during writing, can be interfaced with standard open collector TTL drivers. The data output pin is TTL compatible.

IV. ARRAY PROCESSING

The MNOS memory array is manufactured by an extension of a typical p-channel MOS process. Isolation between the MNOS memory transistor array and peripheral circuitry is achieved by a p-diffused wall through an n-type epitaxial layer into a p-type substrate. Another diffusion is added to provide n$^+$ contacts to the isolated n regions. The MNOS memory transistor utilizes a stepped-gate structure which results in fixed threshold devices that have a nitride mem-
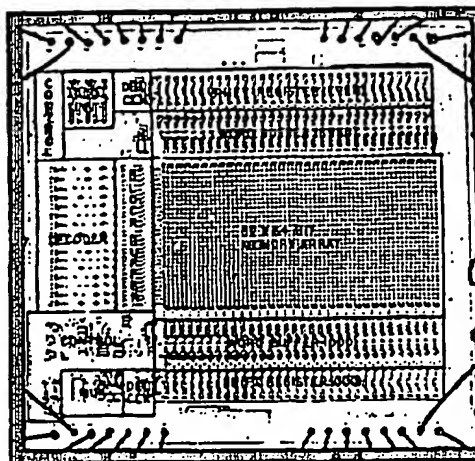
Fig. 12. Memory chip photograph.

ory layer as the upper part of the gate dielectric. This does not cause any problems in fixed gate stability. Eight masking steps are necessary to manufacture this array. A single layer metallization of aluminum is used. The layout rules are relatively generous, with an average of 0.4 mil widths and spacings used. In the design approach static (resistance ratio) logic is used. This requires care in processing to meet performance requirements.

V. PERFORMANCE CHARACTERISTICS

Over 2000 fully functional chips have been fabricated and tested. Of these about 500 have been utilized in hardware which is operational. This limited production effort has provided the opportunity to determine major yield-limiting factors and to achieve a uniform controlled yield of the device. The experience and data obtained have demonstrated the producibility of devices utilizing the MNOS technology, and the viability of the technology for use in complex systems.

The testing of these devices has provided considerable data on various key parameters. One of the most important parameters is the read voltage window. This is the range of read voltage levels at a given time for which data can be correctly assessed. A typical memory array read voltage characteristic is shown in Fig. 13. The lower window edge corresponds to the low conduction (high V_T) setting on the worst case (i.e., smallest high threshold) of the 2048 bits. A nearly constant slope is followed on the lower edge. The upper high conduction (low V_T) edge is flat for the first decades of time but then assumes a constant slope. This upper edge flat portion occurs because the stepped-gate memory transistor is a standard variable threshold transistor in series with a fixed (non-shifting threshold) device. The entire structure has a mini-
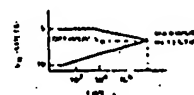

Fig. 13. Typical memory array read voltage characteristics.

mum threshold equal to that of the fixed device, even though the threshold of the variable device is initially smaller. Until the variable threshold relaxes to a setting less than the fixed portion minimum, it does not determine the threshold of the memory transistor. The linear slopes at later times characterize the behavior of the memory structure and permit extrapolation of an ultimate retention limit. End of data storage clearly occurs when the window edges intersect. Thus, an optimum read voltage level can be defined for a given chip. Similarly, optimum retention and read voltage can be selected for groups of chips. The distribution plot in Fig. 14 shows the number of memory arrays that are fully functional at 0.2-V increments of read voltage [4]. The initial window and window after 10^5 reads are plotted from actual measurements. The 10^{12} reads plot is extrapolated on the assumption of a linear read voltage change with the logarithm of time. Several conclusions can be drawn from Fig. 11. The optimum read voltage for the lot is about -7.8 V. Also, long retention requirements have some yield impact. The most significant feature is the shape of the curve, which indicates predictability and process consistency since devices from about 100 batches fabricated over several months' time are represented.

Fig. 15 shows the read voltage range as a function of time at both 25°C and 125°C under a power-off condition for typical
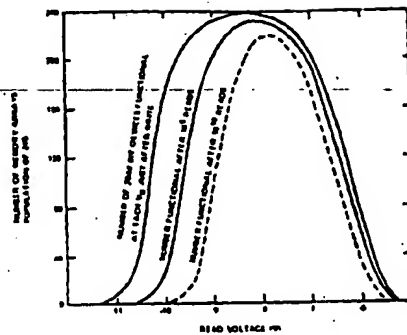
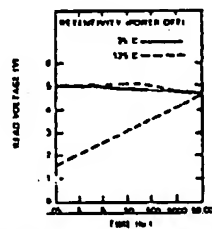Fig. 14. Distribution of V_R settings for which worst bit of 2048 bits per memory array is functional.

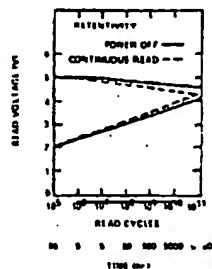Fig. 15. Retentivity (read voltage versus time) for power-off condition.

Fig. 16. Retentivity (read voltage versus time) for continuous read.

devices. This graph indicates a retentivity (time which data may be stored in the array while still providing an adequate read voltage range) to provide reliable system operation exceeding one year at 25°C and approaching one year at 125°C.

Fig. 16 indicates the effect of continuously reading a single memory cell. This condition is generally more severe than a power-off condition, the reason being that application of a read voltage to the gate of the MNOS cell is like a "write"

operation at a smaller voltage. The effect of this is a reduction in retentivity of about one-half order of magnitude of time.

VI. SUMMARY

A 2048-bit MNOS BORAM memory chip, used as the storage element in two BORAM memory systems, has been described. These systems represent a major milestone in the development and application of MNOS memory technology. Data obtained from operating systems indicate that difficult performance requirements have been met and that MNOS memories can be fabricated in acceptable yields to function in large complex memory systems.

ACKNOWLEDGMENT

The authors would like to acknowledge the contributions made by B. Peterson and A. Ventresca for the design and layout of the memory array, and to M. O'Connell for his work in developing and improving the test capability. We are also grateful to Dr. R. Newman for his trust and guidance.

REFERENCES

[1] R. F. Spencer, "Semiconductor block-oriented read write memory," U.S. Patent 3 898 632.
[2] C. A. Beltz, "MNOS BORAM memory development," in GOMAC Dig., vol. V, June 1974, pp. 20–21.
[3] F. A. Sewell, H. A. R. Wegener, and E. T. Lewis, "Charge storage model for variable FET memory," Appl. Phys. Letters, vol. 14, Jan. 1969, pp. 45–47.
[4] R. J. Lodi et al., "Chip and system characteristics of a 2048-bit MNOS-BORAM LSI circuit," in 1976 Int. Solid-State Circuit Conf., Dig. Tech. Papers, p. 62.

Millicent B. Borovicka received the B.S. degree in chemistry from Simmons College, Boston, MA, in 1963.

In 1964 she was an Engineer with Sylvania Electric Products, working on semiconductor device failure analysis. She joined the Sperry Research Center, Sudbury, MA, in 1964 and has since been made a member of the Technical Staff. She has worked on MOS transistors and CCD's, but her main concern has been with the process control of MNOS memory transistors and the manufacture of MNOS-LSI circuits. She is currently in charge of all wafer fabrication at the Sperry Research Center.

Bernard B. Kosicki (M'73) received the B.A. degree from Wesleyan University, Middletown, CT, in 1961, and the M.A. and Ph.D. degrees from Harvard University, Cambridge, MA, in 1962 and 1967, respectively.

He was a member of the Technical Staff at Bell Laboratories, where he worked on thin-film dielectrics and compound semiconductors, silicon devices, and CCD's. He joined the Sperry Research Center, Sudbury, MA, as Department Manager of Pilot Line Operations. In this capacity he was intimately involved in improving the yield of MNOS-LSI circuits. Since 1976 he is with General Instruments, Hicksville, NY.

Robert J. Lodi was born in Springfield, MA, in 1945. He received the B.S. degree in electrical engineering from Worcester Polytechnic Institute, Worcester, MA, in 1966, and the M.S. degree in electrical engineering from Northeastern University, Boston, MA, in 1970.

He has worked as a Senior Designer and Test Equipment Development Engineer at Sylvania Electric Products, Inc., in Woburn, MA, and at Viatron Computer Systems, Inc., in Bedford, MA. In 1970 he joined the Sperry Research Center, Sudbury, MA, where he is a member of the Technical Staff engaged in the design, layout, and test development of LSI circuits.

Thomas A. Pogemiller (S'68–M'70) was born in Minneapolis, MN, on October 11, 1948. He received the B.E.E. degree and B.S. degree in business from the University of Minnesota, Minneapolis, in 1970 and 1974, respectively.

Since joining Sperry Univac Defense Systems Division, St. Paul, MN, in 1972 he has been involved in design and evaluation of MOS and MNOS LSI memory devices. He is currently a Lead Designer in the Complex Array Development Group, working on mass memory and random-access MNOS arrays.

H. A. Richard Wegener (M'66–SM'75) received the B.S. degree from Columbia University, New York, NY, in 1951, and the Ph.D. degree from the Polytechnic Institute of Brooklyn, Brooklyn, NY, in 1965.

He has been in the semiconductor field since his employment at Tung-Sol Electric Research Laboratories in 1955, where he worked on germanium alloy transistors, junction field-effect transistors, and RTL circuits. He joined the Sperry Research Center, Sudbury, MA, in 1964, where he is now a Department Manager of Integrated Circuit Development. He has worked on MOS transistors and CCD's, but since 1966 his main concentration has been on MNOS memory transistors, especially on aspects involving device theory, structure and processing, MNOS-LSI circuit design and processing, and radiation hardness.

Marshall W. Eklund was born in Duluth, MN, on March 21, 1945. He received the B.E.E. degree from the University of Minnesota, Minneapolis, MN, in 1967.

In 1967 he joined Sperry Univac Defense Systems Division, St. Paul, MN, and became involved in the design and development of two computer test vehicles that were forerunners to company products. He has been Project Engineer for several programs including a CMOS computer, MOS mainframe memory, and a medium-scale hybrid ceramic technology computer. Since 1974 he has been responsible for the development of several MNOS memory systems.

An On-Chip Smart Memory for a Data-Flow CPU

GREGORY A. UVIEGHARA, STUDENT MEMBER, IEEE, YOSHINOBU NAKAGOME, MEMBER, IEEE,
DEOG-KYOON JEONG, STUDENT MEMBER, IEEE, AND DAVID A. HODGES, FELLOW, IEEE

Abstract—Register Alias Table (RAT) is a smart memory that is embedded in HPSm (High-Performance Substrate), a Berkeley data-flow CPU. It is a multiport memory that has content addressability and support for branch prediction and exception handling, in addition to conventional READ and WRITE operations. An experimental 1240-bit smart memory chip is implemented in a 1.6-μm double-metal-scalable CMOS process. This memory performs 15 operations within a cycle time of 100 ns, has 34 658 transistors, occupies an area of 3.8 mm × 5.3 mm, and dissipates 0.51 W [1], [2].

I. INTRODUCTION

DUE TO advances in fabrication capabilities, the density of conventional memories has increased to the extent that 1- and 4-Mbit DRAM's are already in mass production and announcements have been made of experimental 16-Mbit DRAM's [3]. The increased integration has also given rise to a parallel trend where logic combined with data storage elements generates less dense but more functional memories. Video RAM's (VRAM's) and content addressable memories (CAM's) are examples of such more functional or so-called "smart" memories. VRAM's combine conventional RAM's with serial access registers and some control logic to support bit-mapped graphics display systems [4]. CAM's combine conventional RAM's with XOR comparators to perform parallel data search without extensive address handling for supporting parallel processing, expert systems, and artificial-intelligence applications [5]–[7].

Smart memories are required to satisfy the demands by systems designers for more versatile systems that are im-
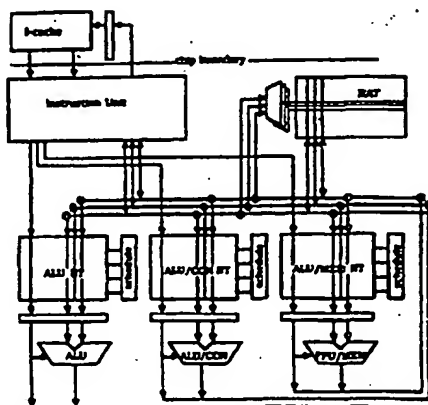
Fig. 1. HPSm CPU block diagram.

plemented directly in silicon. For instance, HPSm (High-Performance Substrate) is a data-flow CPU that uses hardware to control out-of-order execution by employing on-chip smart memories [1], [8], [9]. Fig. 1 shows the HPSm block diagram. HPSm uses data-flow techniques to coordinate out-of-order execution. The out-of-order execution model gives it high throughput since instructions whose operands are not ready do not block subsequent ones. To avoid the complexity of a centralized control of out-of-order execution, it uses a decentralized control approach where the control is embedded in the on-chip memories, making them "smart." This "smartness" poses significant circuit design challenges. Because of the extra complexity of these smart memories, test chips are designed to study them one by one before they are incorporated in the HPSm data-flow CPU. This paper deals with the experimental Register Alias Table (RAT) smart memory chip.

RAT is the main smart memory on the HPSm CPU chip and plays the major role of manipulating the data-flow graph, the central data structure in any data-flow CPU.
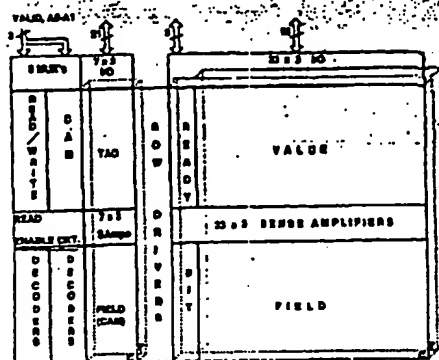
Fig. 2. Memory architecture.

Fig. 3. RAT microphotograph (5.37 mm × 2.97 mm).

It is analogous to a register file in a conventional von Neumann CPU but with extra capabilities for enhanced data manipulation and support for out-of-order execution control. In Fig. 1, RAT communicates with three other smart memories (MEMORY NODE TABLE, ALU NODE TABLE, and ALU/CONTROL NODE TABLE) through three ports. These smart memories in turn support three CPU function units (MEMORY, ALU, and ALU/CONTROL). RAT has a content-addressable tag field to support associative operations, and two backup copies per data element to support branch prediction and exception handling.

II. MEMORY ARCHITECTURE

A. Overall Architecture

The memory architecture is shown in Fig. 2. Information is stored in a 31-word format (40 bits/word) forming a 1240-bit array that is accessible from the external world. The core is partitioned into three fields. The 7-bit tag field (a 2-bit non-content-addressable tag field and a 5-bit content-addressable tag field) is used to tag the 32-bit data in the value field. The ready bit field indicates if the data are valid. Each field is divided into two halves by a set of sense amplifiers. Each of the other three smart memories in Fig. 1 logically sees one I/O port looking into the RAT which it uses repeatedly for all its interaction with the RAT. By multiplexing each port for two READ's, one WRITE, and two associative WRITE's per cycle, the RAT enjoys the benefits of a 15-port memory while paying the price of a three-port memory in terms of area and power. (However, the time-sharing of the ports by different operations certainly has an impact on the cycle time.)

Six multiplexers and six decoders are used — three for conventional READ/WRITE operations and three for assisting the associative WRITE operations. The read-enable circuitry divides the decoders into an upper half and a lower
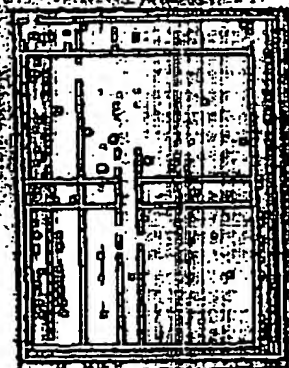
half corresponding to the upper and lower halves of the core, respectively. This circuitry is used to conditionally inhibit some ports during READING. The row drivers buffer and multiplex the tag word line (for conventional operations) and the tag match line (for content addressable operations) to drive the ready bit/value field word lines.

The tag and ready bit fields have one backup copy while the value field has two backup copies to support branch prediction and exception handling. So indeed, although only 31×40 bits are directly accessible from the external world, the core actually stores 3470 bits. The programmer-visible copy of the memory is called the current copy (shown as C in Fig. 2) while the first and second backup copies are called the transit (T in Fig. 2) and settled (S in Fig. 2) copies, respectively.

B. Word-Line/Bit-Line Organization

Conceptually, the RAT can be viewed as a three-dimensional memory that is 31 words long, 40 bits wide, and three backup copies (the boxes in dotted lines in Fig. 2) deep. However, since a nonplanar technology (e.g., optical [10]) is not available for implementation, this "three-dimensional" memory is physically realized as a "two-dimensional" memory using a planar technology as shown in Fig. 3.

Fig. 4 depicts how the word lines and bit lines are organized. For clarity, only the fifteenth and sixteenth words are shown. The word-line organization is as follows. For each word, three tag current word lines (tag-C.WLs) are driven by the READ/WRITE decoders (Fig. 3) to support three-port conventional READ and WRITE operations. The CAM decoders (Fig. 3) drive three tag current match lines (tag-C.MLs) and three tag backup match lines (tag-B.MLs) for each tag word. The row drivers multiplex the three tag-C.WLs and the three tag-C.MLs to drive the three value current word lines (val-C.WLs) of each ready/value
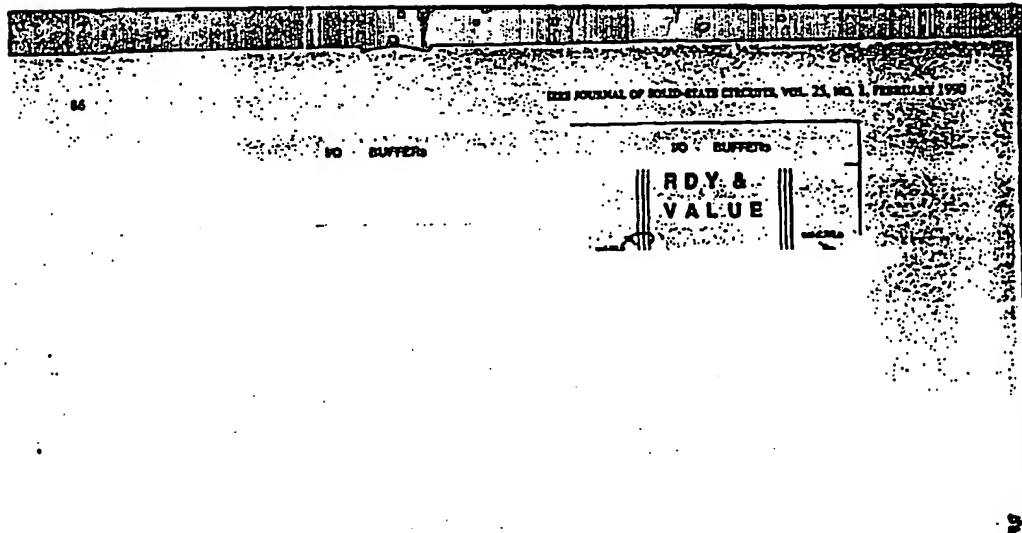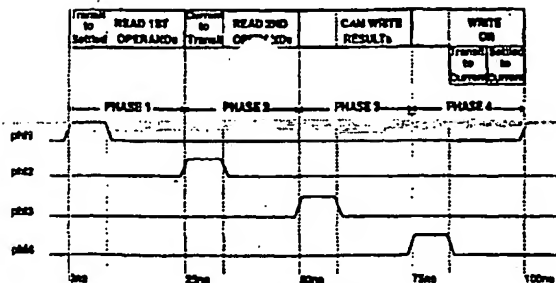
86

Fig. 4. Word-line/bit-line organization.

Fig. 5. Timing sequence.

word. Also, the row drivers receive the three tag-B.MLs as inputs to drive the three value *backup* word lines (val-B.WLs). The bit lines are organized as follows. Three single bit lines are used for the two non-CAM tag bits and the ready/value bits to minimize area. To provide complete logic comparison, it is necessary to use three bit-line pairs for the CAM tag bits. The *C.cells*, *T.cells*, and *S.cells* for each word are laid out side by side as shown in Figs. 3 and 4.

Because a planar technology is used, the area, word-line and bit-line capacitances, power, and cycle time are increased. The area is increased due to the multiplicity of bit lines, word lines, and backup copies. The word-line capacitance is increased since the word line has to traverse at

least three bit lines for each bit. The bit-line capacitance is increased since the bit line has to traverse the backup copies and several word lines and match lines for each word. The increased capacitances have a severe impact on the power and cycle time. These difficulties necessitate a very careful choice of the data cells. The cells used to minimize the severity of these problems are discussed in Section IV.

III. BASIC OPERATIONS

The timing sequence is illustrated in Fig. 5. All signals are aligned to the four-phase clocks of the data-flow CPU as shown in the figure. In the sequel, phase *I* is defined as

the time segment from the rising edge of ϕ_i to the rising edge of ϕ_{i+1}. In each of the first two phases, three words from three (possibly different) locations are read out into the three different ports. Then in phase 3, computed results from the three function units are written into the *current* and *transit* cells of the ready/value fields if the interrogative tags match the stored tags. Phase 4 is used to update the data-flow graph by writing new values into the tag and ready fields.

The RAT supports branch prediction and exception handling by allowing SAVE and REPAIR operations. These operations are defined as follows. If a conditional branch is encountered in the instruction stream, the data-flow CPU predicts the branch to be taken and informs the RAT to save (the SAVE operation) its *current* data in the *transit* cells by raising the C2T signal in ϕ_2. If the CPU later on discovers that the prediction was incorrect, it informs the RAT to recover (the REPAIR operation) the last saved data by raising the T2C signal in phase 4. This has the effect of copying the *transit* data into the *current* cells. Exception handling needs all the SAVE/REPAIR functions of branch prediction [8]. It also requires that the *transit* data be copied into the *settled* cells (in ϕ_1 with T2S) and that the *settled* data be copied into the *current* cells (in phase 4 with S2C) [8].

IV. CIRCUITS

A. Cells

1. Value "Triple Cell": The schematic for the value cell is shown in Fig. 6(a) while the microphotograph is in Fig. 6(b). It has an area of 135 μm $\times$ 58 μm. The value cell has three data storage elements: one pseudo-static cell for the C.cell, one dynamic cell for the T.cell (for branch-prediction support), and one dynamic cell for the S.cell (for exception handling support). The READ/WRITE operations are: a one-port (one of three available ports) conventional WRITE to the C.cell, a three-port conventional READ from the C.cell, and a one-port (one of three available ports) associative WRITE to the C.cell and the T.cell. The SAVE/REPAIR operations are: a current-to-transit SAVE, a transit-to-settled SAVE, a transit-to-current REPAIR, and a settled-to-current REPAIR.

The C.cell is implemented as a pseudo-static cell for two reasons. The first reason has to do with the high band-width requirement of the RAT. To save area, the traditional "bit-line pair" cannot be used to implement the three-port RAT. Instead, a "single-bit-line" approach has to be used. Although the single-ended cell used by Stewart and Dingwall [11] is compact, it was rejected to avoid the complexity of using boosted word lines. READ and WRITE operations are performed on the C.cell by directly controlling the feedback path in the cell. The *REFRESH* signal is kept high during a READ operation while it is taken low for WRITE operations. Keeping the *REFRESH* signal high protects cell data during READING while taking it low allows a contention-free WRITE. Therefore, the C.cell be-
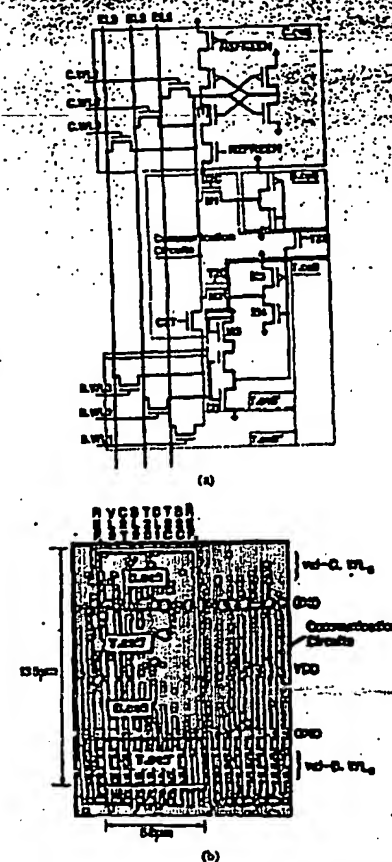

(a)

(b)

Fig. 6. (a) Value "triple-cell" schematic. (b) SEM microphotograph of the value cell after stripping the passivation layer.

haves like a static RAM cell for READING and like a dynamic RAM cell for WRITING. Since the bit lines are shared by the C.cell and the T.cell to save area, there is a grave danger of a sneak path between nodes (1) and (2). The sneak path is blocked by ensuring that the word lines of the T.cell are low throughout the READ operation. In conclusion, the pseudo-static technique permits the implementation of the RAT as a three-port memory with three single bit lines rather than three bit-line pairs with substantial savings in area, but without paying the price of using boosted word lines.

The second reason for using the pseudo-static cell has to do with the REPAIR operation. The REPAIR operation is

essentially a WRITE operation. But a WRITE operation is easier to carry out with large peripheral circuitry than with cell circuitry if the memory complexity (area, power, etc.) must not be prohibitively large. This is because the WRITE buffer increases the complexity of the memory by $O(n)$ (where n is the memory dimension) since the WRITE buffer is shared by all the cells of each column, while any REPAIR circuitry increases the complexity by $O(n^2)$ since it must exist in each cell of the array. If a static cell had been used, there would have been a lot of contention during a REPAIR operation from the T.cell or from the S.cell. First, this contention means that the T.cell, the S.cell, and the REPAIR pass transistors ($M1$ and $M2$) have to be large for the REPAIR operation to be successful. Second, the contention leads to high power dissipation and high peak currents with the attendant inductive noise problems. The need to carry out the REPAIR operation in all the cells of the core (992 cells) at once compounds the above two contention-related problems. In the RAT value cell, the REPAIR operation is carried out by keeping the REFRESH signal low while T2C or S1C is raised high. In this way, the contention between the current and backup parts of the cell is avoided.

Only one of the backup cells (the T.cell) interacts with the external world. Dynamic cells are used in the backup section for area and simplicity reasons. (A pseudo-static version had to be used in the current part to avoid the complexity of providing REFRESH.) The use of dynamic cells for the backup section also prevents contention during the SAVE operations. In the T.cell, a full ONE cannot be written because all the pass transistors attached to node (Z) are NMOS transistors. This problem, which is exacerbated by the body effect, leads to static power dissipation. The $M3$ transistor is used to minimize this problem. The $M4$, $M5$ inverter is necessary for logical correctness. The dynamic S.cell does not need REFRESH since the CPU design guarantees that its value is used within 1 ms. The REFRESH for the T.cell is initiated by software with no hardware cost. Normally, after an average of five instructions, branch prediction is made and the T.cell is written into from the C.cell. If a branch prediction has not taken place in a reasonable time (1 ms, say), the compiler inserts an artificial branch in the code to force a current-to-transit SAVE, thereby REFRESHing the T.cell.

2. Tag "Double-CAM Cell": The schematic for the tag cell is shown in Fig. 7(a) while the microphotograph is in Fig. 7(b). It has an area of 135 μm × 88 μm. The tag cell has two data storage elements: one pseudo-static cell with three tag comparators for the C.cell and one dynamic cell with three comparators for the T.cell (for branch-prediction support). The READ/WRITE operations are: a one-port (one of three available ports) conventional WRITE to the C.cell and a three-port conventional READ from the C.cell. The associative operations are: a three-port tag comparison with the C.cell and the T.cell. The SAVE/REPAIR operations are: a current-to-transit SAVE and a transit-to-current REPAIR.
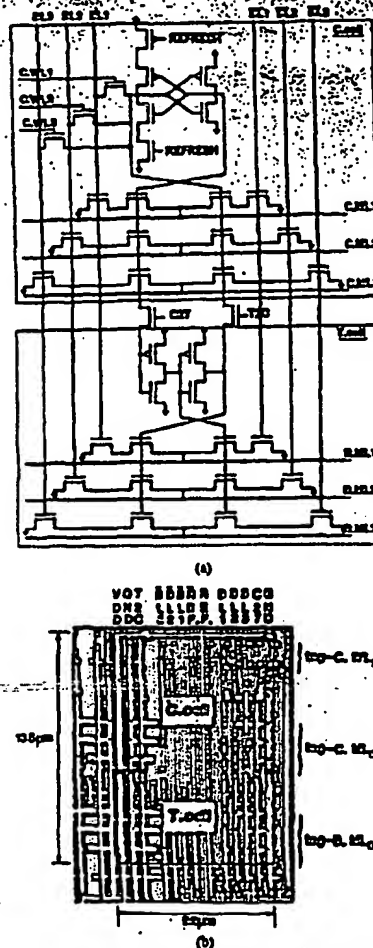

(a)

(b)

Fig. 7. (a) Tag "double-CAM-cell" schematic. (b) SEM microphotograph of the tag cell after stripping the passivation layer.

The pseudo-static cell was used to implement the tag C.cell for the same reasons that justified its use for the value C.cell. The tag cell had to be implemented with bit-line pairs to allow a complete logic comparison with an external tag. (Fortunately, the total area cost is bearable since only five CAM tags are needed.) The tag CAM cell needs six tag comparators so that both the C.cell and the
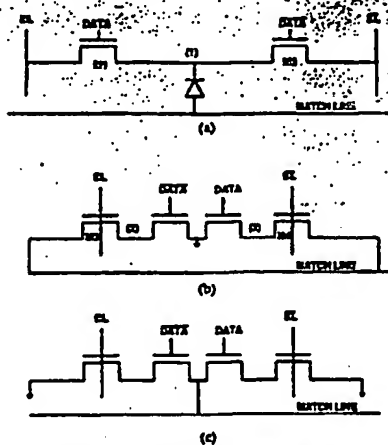
Fig. 9 (a) Mundy *et al.* CAM cell comparator. (b) Kadota *et al.* CAM cell comparator. (c) Proposed CAM cell comparator.
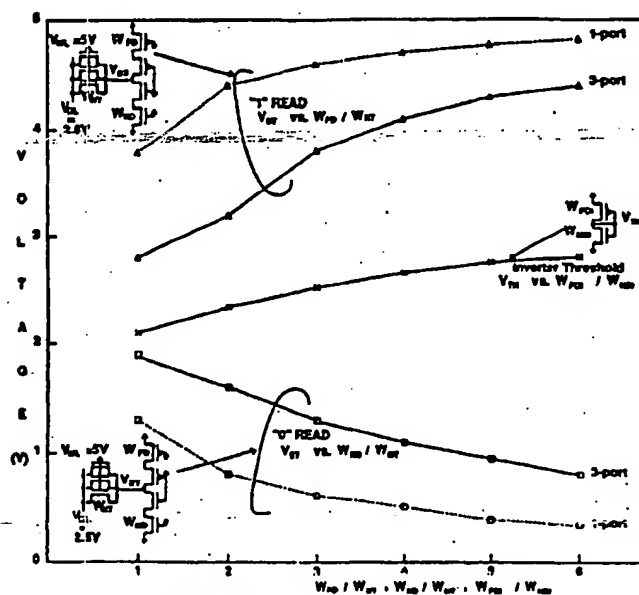

Fig. 9. Noise margin dc analysis for *current* cell (nominal).

A 128K × 8 70-MHz Multiport Video RAM with Auto Register Reload and 8 × 4 Block WRITE Feature

RAY PINKHAM, MEMBER, IEEE, DONALD RUSSELL, ANTHONY BALISTRERI, TROY H. HERNDON, DANIEL ANDERSON, ASWIN MEHTA, THANH NGUYEN, NGAI HUNG HONG, MEMBER, IEEE, HIROSHI SAKURAI, SEISHI HATAKOSHI, AND ANDRE GUILLEMAUD

Abstract —An 80-ns 1-Mbit multiport video RAM (VRAM) can be organized as 128K × 8 or 256K × 4. Uninterrupted serial data streams of 70 MHz are achieved by combining pipelining and interleaving techniques with an internally triggered automatic memory-to-register transfer mechanism. DRAM bandwidth is enhanced by a block WRITE feature which can write to many as four column address locations in every CAS cycle. The WRITE-per-bit feature has been expanded by including an on-chip WRITE-per-bit latch and an extended mode of operation to simplify its use in a wider range of systems.
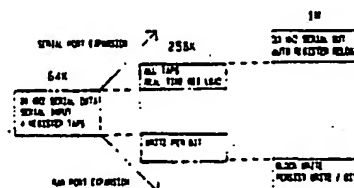
I. INTRODUCTION

THE MULTIPORT video RAM (VRAM) was introduced in 1983 [1]. It featured a 64K × 1 random access port coupled to a 256 × 1 serial access port for providing data to a graphics display. In 1985 256K devices were introduced [2] and were organized as 64K × 4. WRITE per bit and real-time data transfer from memory to serial register represented enhancements over the 64K device and became standard features for VRAM's. VRAM devices at the 1-Mbit density organized as either 128K × 8 or 256K × 4 suffer from the fact that they are two to four times deeper than 64K devices, yet the data in the RAM must still be processed and updated in the same general time period. The need exists for higher bandwidth on both the serial and random access ports to achieve low-cost, high-resolution graphics display systems. The 1-Mbit VRAM described herein can be organized as either 256K × 4 or as 128K × 8 and contains features that can provide this higher bandwidth. Fig. 1 shows the evolution of VRAM devices from the introduction of the 64K VRAM to current 1-Mbit devices.

Fig. 1. Evolution of VRAM's.

II. PIPELINED SERIAL ACCESS OPERATION

The device features both serial input and serial output operation. During serial output mode, 70-MHz uninterrupted serial data streams are achieved by combining a pipelined and interleaved architecture with an internally triggered automatic memory-to-register reload feature. Fig. 2 shows a block diagram of the serial architecture. The pipeline is divided into two stages as shown. The first stage contains the serial counter and decoder which select the next two bits (even and odd address) to be output to the serial data lines (designated SE and SO). The serial data lines are the inputs to the second stage of the pipeline. The second stage of the pipeline is interleaved, accepting both the even and odd bits from the serial data lines into an isolation latch within the first I/O MUX, and then alternately selecting first the even and then the odd bit for serial output on successive serial clock cycles. While the odd bit is selected for output, the first stage of the pipeline decodes the next even/odd pair and transfers their data to the serial data lines. The data arrive at the first I/O MUX but are prevented from overwriting the previous data in the isolation latch by pass enable PSN being low. On the subsequent even clock pulse, PSN goes high and the data bits flow into the latch with the even bit flowing directly to the output. PSN is then reset by the falling edge of the
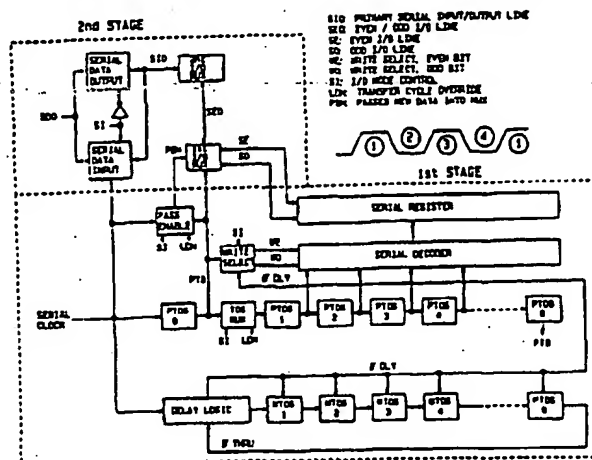
1134

Fig. 1. Block diagram of the serial architecture.

even clock pulse. Included in Fig. 2 is the serial clock waveform describing the operation of the pipeline.

The serial counter is a 9-bit ripple-type counter. Since the serial access is pipelined, there is no need for the extra logic required to construct a slightly faster synchronous counter. The counter is assembled by connecting the complement output of each stage to the true clock input of the next stage. The toggle bits operate in two phases. Both true and complement outputs from the previous stage are fed to the inputs of the toggle bit, except for PTOG 0 which has the serial clock as inputs. Fig. 3 shows the schematic diagram of the serial counter bit. The first phase of the toggle operation begins when a low-to-high transition occurs on the true output of the preceding stage. This loads the master portion of the toggle bit from the toggle latch and sets up the bit to toggle. On the high-to-low transition, the load of the master is disabled and the toggle latch is overwritten by the complement input going high and pulling one side of the slave to ground.

The pipeline is controlled by operating the least significant bit (PTOG 0) of the serial counter 180° out of phase with respect to the other bits in the counter. This is accomplished by reversing the true and complement outputs of PTOG 0 at the input to PTOG 1 via the toggle MUX. The toggle bits will toggle when a low-to-high transition occurs at the clock input. Thus, when an odd address is selected, PTOG 0 will go high at the input to PTOG 1 because of the toggle MUX, and the counter will increment on the odd address and prefetch the next two bits. Fig. 4 shows an oscillograph of the 70-MHz performance under typical operating conditions with one cycle expanded to highlight serial access time.
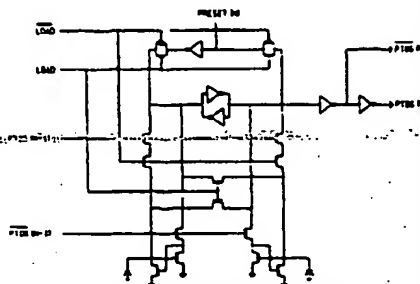

Fig. 3. Schematic diagram of serial counter bit.

In serial output operation pipelining is responsible for doubling the data bandwidth. In serial input operation, however, pipelining creates difficulties in synchronizing the input data to the proper serial address. For this reason, the pipeline is defeated by "unreversing" the outputs of PTOG 0 via the toggle MUX. Nearly the same bandwidth is achieved as in serial output mode since the critical path to write a bit in the data register is roughly the same as the first stage of the pipeline when operating in serial output mode.

Moreover, pipelining introduces difficulties when loading the data register from the memory array while maintaining continuous serial output. When the new data are loaded into the data register, the new start address is also loaded. The new bit must then be sent directly to the
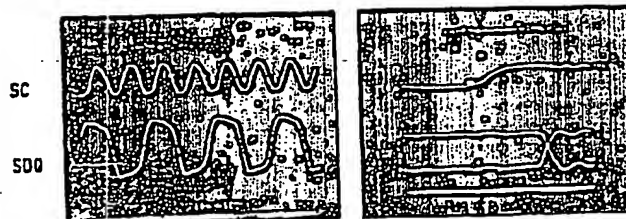
Fig. 4. Oscillograph of 70-MHz multiport VRAM.

output since the second stage of the pipeline is empty. In order to resolve this, the pipeline is defeated during the first serial clock cycle after the memory-to-register transfer and the first two bits of data flow all the way into the isolation latch with the proper bit being read at the output. Since the address for the first bit comes from a tap address latch which presets the counter, there is no time lost waiting for the counter to ripple to the next address, making the 70-MHz data stream truly continuous.

Fig. 2 also shows a "mock" toggle counter with corresponding delay logic. Since the normal serial counter is of the ripplethrough type, spurious outputs from each stage during the ripplethrough could cause the decoder to activate the wrong address momentarily. During serial input mode, this could cause data to be written to the wrong location. To remedy this, the MTOG counter contains toggle bits which duplicate the delay of the PTOG counter. The MTOG counter is preset to all ONE's before each pertinent rising edge of the serial counter to simulate the worst-case ripple through the PTOG counter. WRITE enable to the serial data register is defeated until the ONE-to-ZERO transition is detected from the highest order mock toggle bit. MTOG 8. This indicates that the PTOG address has become valid.

III. AUTO REGISTER RELOAD OPERATION

Earlier VRAM devices featured "real-time data transfer" or "midline load" from memory to register. This feature is intended to permit a continuous serial data stream during the reload of the serial data register from the memory array. During high serial frequency operation, the use of this feature becomes extremely difficult to implement. This is due to the tight timing constraints during the reload cycle, particularly between the serial clock and the transfer enable (TRG) control inputs. Fig. 5 shows the traditional method of reloading the data register with continuous serial data output. The rising edge of TRG initiates the actual data transfer in the midline-load cycle. It must occur at the proper time between successive serial clock pulses in order to transfer the data to the serial register and properly synchronize the new data to the appropriate serial clock. To rectify this problem, this device includes an automatic register reload feature which internally detects when the last bit in the data register has been output and
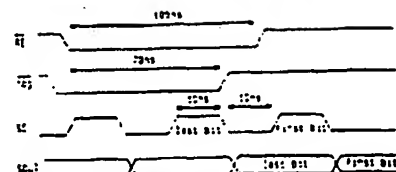

Fig. 5. Traditional register-to-memory transfer timing.
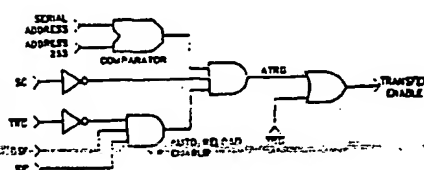
Fig. 6. Logic and timing for auto reload feature.

asserts its own internal transfer signal to allow new data to be loaded into the data register. Fig. 6 shows the logic and timing for the auto reload feature. Some time previous to accessing the last bit in the serial data register, the auto register reload cycle is initiated. Control is similar to standard VRAM's except that an additional special function pin, DSF, has been added and is held high on the falling of RAS enable (RE) to distinguish auto register reload from normal register reload. When the auto register reload condition is detected, the VRAM sends a transfer

Fig. 7. Operation of block WRITE mode.

(a)

(b)

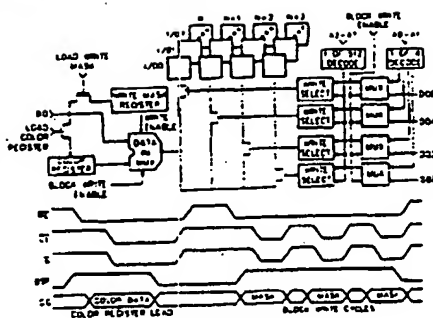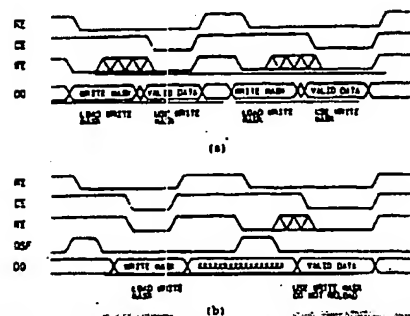Fig. 8. WRITE-per-bit comparison: (a) conventional method, and (b) improved method.

TABLE I
SPECIAL FUNCTION TRUTH TABLE

Fig. 9. Special function block diagram.

busy handshake signal ($\overline{XFB}$) back to the processor to block all memory cycles until the transfer has been completed. The rising edge of $\overline{RE}$ is also defeated internally until the transfer has taken place. The $ATRG$ pulse is synchronized to the internal operation of the serial clock circuitry, allowing it to occur at the optimum time between clock pulses and eliminating the need for external control. Once the transfer has occurred, $\overline{XFB}$ goes back high and control is returned to $\overline{RE}$.

IV. BLOCK WRITE OPERATION

To facilitate faster updating of the bit-map memory through the random access port, this device is equipped with an 8×4 block WRITE mode. This feature allows color fill patterns to be written to multiply memory address locations in every column-address cycle. Fig. 7 describes the operation of the block WRITE mode. An 8-bit data pattern is loaded into an on-chip register using standard DRAM WRITE cycle timing but holding the special function pin (DSF) high on the falling edges of $\overline{RE}$ and $\overline{CE}$. This signals to the device that the destination for writing
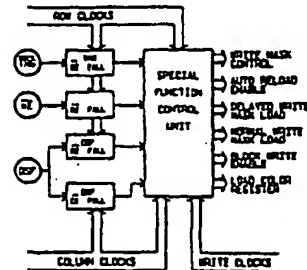
the data on the DQ pins is to be the color register instead of an address in memory. During future WRITE cycles, the user can choose between normal WRITE and block WRITE operation by holding the DSF pin low or high, respectively, on the falling edge of $\overline{CE}$. If block WRITE mode is selected, the contents of the color register can be written to any subset of four contiguous column address locations in memory. A 4-bit address mask is applied to the even DQ pins on the falling edge of the later of $\overline{W}$ or $\overline{CE}$. These four bits replace the two least significant column addresses $A0$ and $A1$. The six high-order addresses $A2-A7$ select a group of four contiguous columns. The mask data applied to the DQ pins act as individual WRITE enables for each of the four selected columns: $DQ0$ being high enables the WRITE to the lowest order column, $DQ2$ being high enables the WRITE to the next column, $DQ4$ to the next, and $DQ6$ to the highest order column. Any combination of the four columns can be written with the contents of the color register. The block WRITE can also be used with the WRITE-per-bit feature to permit masking by memory plane as well. Thus, a fourfold improvement in bandwidth can be achieved during color fill operations such as window clears and polygon fills.
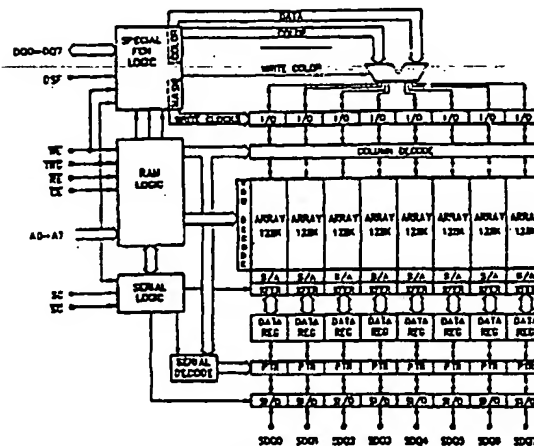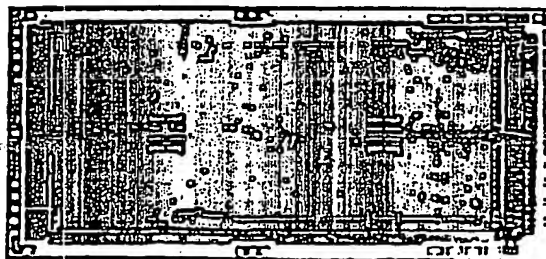
Fig. 10. Block diagram of entire chip.

Fig. 11. Die photo of the device.

TABLE II

Organization	128K x 8 DRAM
	256 x 8 Serial
Technology	CMOS, triple metal, double level
	poly/polycide, single level
	metal, trench capacitor cell
Design Rules	8 um
Cell Size	2.5um x 8.5um
Die Size	5.3mm x 13.5mm
Row Access Time	83ns
Serial Access Time	10ns
Row (reductive data ports)	62ns
tow - ins/transdata	7.5ns

V. EXTENDED WRITE-PER-BIT OPERATION

Previous generations of VRAM's incorporated a WRITE-per-bit feature for writing to selective data inputs of the VRAM device while defeating the WRITE operation to the other(s). One consequence of this feature as implemented on previous devices is that the WRITE mask controlling which inputs are to be written had to be supplied during every WRITE cycle and was strobed into the device on the falling edge of $\overline{RE}$ at the same time as the row address. This made it difficult to utilize the WRITE-per-bit capability in systems with common address and data buses as well as forcing the user to store the WRITE mask for use in multiple WRITE cycles. This VRAM improves upon previous generations of VRAM's by allowing the WRITE mask to be loaded using standard DRAM timing as described in Fig. 8. Loading the WRITE-per-bit mask is accomplished similarly to loading the color register except that DSF is held low on the falling edge of $\overline{CE}$. Once the WRITE-per-bit mask is loaded by either method, it is latched on chip and can be used on multiple memory WRITE cycles. Also, the user is free to choose between using the stored WRITE

mask, storing a new WRITE mask for use during the current cycle, or unconditionally writing to all data inputs, by selective control of $\overline{WE}$ and $\overline{DSF}$ on the falling edges of $\overline{RE}$ and $\overline{CE}$. Table I summarizes all of the special function cycles described herein with their control sequences. Fig. 9 is a block diagram of the logic unit that generates the control signals for the VRAM functions.

VI. TECHNOLOGY

The VRAM is fabricated in a 1-μm CMOS technology using double-level poly/polycide, single-level metal, and trench DRAM storage capacitors for high noise immunity. Table II lists the key physical, technological, and performance features. Fig. 10 is a block diagram of the entire chip showing both the random and serial ports. A die photo highlighting the key areas of the device is shown in Fig. 11.

ACKNOWLEDGMENT

The authors would like to thank T. Yoshimatsu and S. Morinaga for their diligence in processing the wafers: E. Kameyama, Y. Komai, H. Gotoh, and C. Perrin for their assistance in characterizing the devices; and J. O'Hare, K. Hara, D. Buss, and F. Miu for providing the support and guidance without which this device could not be possible.

REFERENCES

[1] R. Pinkham, K. Guttag, and M. Novak, "Video DRAM excels at fast graphics," Electron. Des., pp. 161–171, Aug. 18, 1983.
[2] S. Ishimoto et al., "A 256K dual port memory," in ISSCC Dig. Tech. Papers, Feb. 1985, pp. 38–39.

development organization and is currently working on the 1-Mbit video RAM.

Anthony Balistreri was born in Heidelberg, Germany, in 1962. He received the B.S. degree in electrical engineering from Marquette University, Milwaukee, WI, in 1985. He is presently working toward the Master's Degree in electrical engineering at Rice University in Houston, TX.

He joined Texas Instruments Incorporated in Houston, TX, in 1982 as a cooperative education student where he worked on EPROM, ROM, and static RAM devices. His last work period was spent with the 256K video RAM design group. After graduation, he took a position in the Advanced Memory Development Department where he currently works as a Design Engineer on 1-Mbit video RAM.

Mr. Balistreri is a member of Sigma Phi Delta.

Troy H. Herndon was born in Dallas, TX, on October 1, 1942. He received his education at the University of Texas, Arlington, and at military technical schools.

He joined Texas Instruments Incorporated, Houston, TX, in 1966 and worked in the MOS Section, Semiconductor Research and Development Laboratory. Since then he has been associated with MOS military programs, CMOS, design automation, high-speed MOS static RAM, and multiport memories. He is currently working on 4- and 16-Mbit dynamic RAM devices. His interests are in VLSI chip layout and CAD/CAM development.

Ray Pinkham (S'77–M'78) graduated from the University of Illinois in Champaign-Urbana in 1978 with the B.S.E.E. degree and did graduate study at the University of Houston, Houston, TX, in the area of computer and microprocessor architecture.

He joined Texas Instruments Incorporated MOS Memory Group in Houston, TX, in 1978 where he worked on the development of a 4K VMOS static RAM. From 1979 to 1982 he worked on the design of high-performance NMOS 4K and 16K static RAM's. From 1982 to August of 1988 he was the Design Manager for the first multiport video RAM, the CMOS 256K video RAM, and the 256K×4 and 128K×8 CMOS video RAM's, and was involved with new product definition. In August of 1988 he joined Inova Microelectronics in Santa Clara, CA, where he currently is the MOS Memory Design Manager.

Mr. Pinkham is a member of Tau Beta Pi and Eta Kappa Nu.

Daniel Anderson is from Silver City, NM. He received the Certificate in Electronic Engineering Technology from Albuquerque Technical-Vocational Institute in 1979 and is currently working toward the B.S. degree in electrical engineering at the University of Houston, Houston, TX.

In September of 1979 he joined the MOS Memory Division of Texas Instruments Incorporated, Houston, TX, where he served as a Design Technician in the NMOS static RAM area until 1982. He then worked as a designer of high-speed multiport memories within the DRAM Product Group, and was involved in the development of the first video RAM, the TMS4161, in 1983. He has designed circuitry for the 256K and 1-Mbit video RAM projects, and holds two patents related to video RAM architecture. He is presently a Circuit Designer for the Military Static RAM program.

Donald Russell is from Lexington, KY. He received the B.S. degree in electrical engineering from the University of Kentucky, Lexington, in 1982.

He joined the MOS Memory Division of Texas Instruments Incorporated, Houston, TX, in July of 1982 where he worked as a static RAM and dynamic RAM Product Engineer. In 1984 he served as a Design Engineer on the TMS4161. From 1985 until the present he has worked as a Circuit Designer in the Advanced Memory Development Organization and is currently working on the 1-Mbit video RAM.

Amin Mehta was born in India in 1959. He received the B.E. degree in electrical engineering from Bangalore University in 1983, and the M.S. degree in electrical engineering from Louisiana State University, Baton Rouge, in 1985.

In 1986 he joined Texas Instruments Incorporated, Houston, TX, as a Design Engineer with the 1-Mbit Video RAM Group where he worked on portions of the serial interface circuitry. Since 1987 he has been involved with the design and development of high-speed 256K families of SRAM.
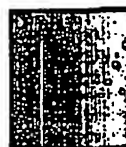
Thanh Nguyen is from Saigon, Vietnam.

He joined Texas Instruments Incorporated in the MOS Wafer Fabrication Group in Houston, TX, in 1980 as a Process Technician. In 1982 he joined the 64K Video RAM Design Group as a Layout Designer. He has since worked on the chip design of the CMOS 256K video RAM and is currently supporting the 1-Mbit video RAM.

Ngai Hung Hong (M'80) received the B.S.E.E. degree from the University of Nebraska in 1974 and the M.S.E.E. degree in material science from the University of Texas at Austin in 1976.

He joined Texas Instruments Incorporated in 1977 as a Circuit Design Engineer working on 64K dynamic RAM devices. From 1975 to 1986 he was responsible for the design and development of the 256K dynamic RAM product family. Currently he is the Manager for the design and development of the 1-Mbit video RAM and the high-speed 256K static RAM for military applications.

Mr. Hong is a member of Eta Kappa Nu and Sigma Tau.

Hiroshi Sakurai was born in Miyagi-Prefecture, Japan, in 1946. He received the B.S. degree in 1968 from Tohhoku Institute of Technology.

He joined Texas Instruments Japan in 1973 as a Process Engineer. He is currently working in the MOS Memory Division as Process Manager at the Miho Plant.

Seishi Hatakoshi was born in Kagoshima-Prefecture in Japan in 1949. He received the B.S. degree from Kyusu Institute of Technology in 1976.

After graduation, he worked in the Bipolar Product Engineering Group at Texas Instruments Japan, Hiji, until he joined the MOS Memory Product Engineering Group, Miho Plant, in 1983. He is currently working as Product Engineering Manager of ASIC memory devices (VRAM and FRAM) covering laser probe to final test.

Andre Guillemaud was born in Victoria, B.C., Canada, in 1962. He received the B.S.E.E. degree in 1984 from Gonzaga University in Spokane, WA. He is currently studying for the M.B.A. degree at Houston Baptist University, Houston, TX.

In 1984 he began working for Texas Instruments Incorporated Microprocessor Division in Houston, TX, and joined the MOS Memory Division in 1986 as a Product Engineer with the Video RAM Group.

A Memory-Based High-Speed Digital Delay Line with a Large Adjustable Length

HANS-JÜRGEN MATTAUSCH, FRED MATTHIESEN, JUTTA HÄRTL,
REINHARD TIELERT, AND ERWIN P. JACOBS

Abstract — A new concept for a digital delay line with an arbitrarily adjustable length is presented. The concept is based on a dynamic three-transistor cell memory with pointer access and offers high operating frequency, large adjustable length, and low power dissipation. The adjustable delay requires only a small overhead for control logic. An experimental chip with 60K transistors, which utilizes this concept, has been built in a 1.5-μm CMOS technology. The adjustable delay ranges from 1 to 4096 clock cycles for a 4-bit-wide data word. Correct operation of the chip has been verified for clock frequencies in the range of 3 kHz to 30 MHz. Therefore the circuit is suitable for audio as well as video applications.

I. INTRODUCTION

IN THE FUTURE more and more fields for the application of modern electronics (e.g., digital communication networks, factory automation, office automation, consumer electronics) will be governed by processing digital data streams. Often these data streams are split into parts, which are processed separately. Afterwards usually synchronization is necessary, because different latencies occur in each processing unit. On the other hand, data must often be delayed deliberately, e.g., for the purpose of building correlations. A delay circuit, which is adjustable within wide boundaries and applicable to a wide frequency range, would be useful for all of these applications.

A recent paper [1] describes the realization of an adjustable delay circuit with a mixed shift register and CCD approach. Application is, however, restricted to the audio range. Another paper [2] describes a memory-based concept. However, the main purpose here is not the adjustable delay, but the easy application to a number of tasks in the field of consumer electronics. Thus the overhead for control circuitry is large and programming of the delay is complicated. Standard circuits which operate up to video frequencies[1] offer only a small programming range and usually have a high power dissipation.

In this paper a memory-based concept for an arbitrarily adjustable, digital delay circuit and an experimental CMOS chip are presented. The basic idea of our solution is to replace the data shifting as in shift registers or CCD solutions by the shifting of a pointer to the word lines of a three-transistor cell memory. Only a small percentage of the data is also shifted in order to ensure simple control circuitry for the programmable delay. We achieve a solution which simultaneously offers a high operating frequency, a wide range of possible delays, and a low power dissipation. The delay of the circuit is simply determined by a binary-coded delay programming word. The experimental 60K-transistor chip can be adjusted in length via a 12-bit programming word to realize any delay from 1 to 4096 clock cycles for a 4-bit-wide data word. Correct operation has been verified in the range from 3 kHz to 30 MHz. Thus the circuit is suited for audio as well as video applications.

II. MEMORY ORGANIZATION

Our circuit concept utilizes a three-transistor memory-cell array, where rows are accessed cyclically by a clock-driven resettable pointer [3].

Fig. 1 shows the three-transistor cell. It has independent word lines for read select and for write select and also independent bit lines for reading and writing. The information is dynamically stored on the gate of a storage transistor. All three transistors are of the n-channel type. A precharge to, say, 5 V is necessary prior to each read access, because the cell can only discharge the read bit line.

The structure of the three-transistor cell allows a READ and a WRITE operation in the same clock cycle. In the first half of a clock cycle data stored in the cell can be read out via the read bit line, and in the second half of the clock cycle new data can be written in via the write bit line, while at the same time the read bit line is precharged to 5 V. Thus high-speed operation of the three-transistor cell memory can be achieved.

Since the read bit line is directly discharged by the cell via the read select transistor, a large output signal is
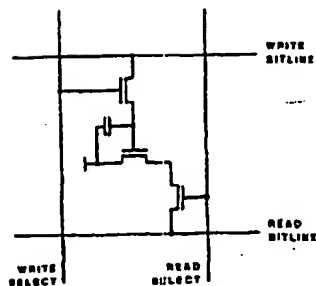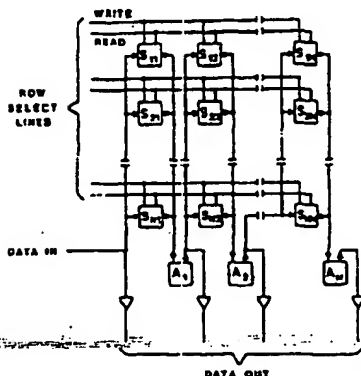
Fig. 1. Three-transistor cell.

Fig. 2. Organization of the memory field. Each S_{ij} represents a memory cell and each A_j an amplifier circuit.

available. This makes simple sensing circuitry possible. The cell can be easily built in a standard CMOS logic process and therefore it can be integrated with other logic circuitry. The memory organization used for the delay-line concept is depicted in Fig. 2. The three-transistor cells are denoted by S_{ij} and are arranged in n rows and m columns. Connections to read and write select lines are drawn on the top of each cell. Connections to the write bit lines and read bit lines are drawn on the left and right side of the cell, respectively. Row select lines run horizontally, bit lines run vertically. Bit-line interconnection is realized in such a way that the read bit line of each column is connected via an amplifier A_j to the write bit line of the following column. Thereby the complete information in an addressed row can be read out in the first half of a clock cycle to be presented at the m data outputs. Then in the second half of the clock cycle it can be written into the same row again, but shifted to the right by one column.

New input data are wired to the write bit line of the first column and therefore are written into the first storage location of the addressed row.

The pointer consists of n dynamic shift-register stages. The row select signals are generated from the intermediate nodes of each shift-register stage. The first two stages of the pointer are depicted in Fig. 3. One additional transistor in each stage serves to reset the pointer. The nonoverlapping two-phase clocking scheme for controlling the pointer is illustrated in Fig. 4. Between the dashed lines a reset operation is assumed to occur. The three additional signals R_0, R_1, and R_2 are necessary for proper control of the reset.

III. CONCEPT FOR THE ADJUSTABLE DELAY

In this section we will explain our concept for the arbitrarily adjustable delay. A 1-bit-wide data word is assumed for the sake of simplicity. The extension to a k-bit-wide data word is straightforward.

Fig. 5 shows a schematic representation of the memory organization, described in the previous section. Cell locations are represented here by squares. These squares are arranged in n rows and m columns. On the left side of Fig. 5 the pointer is indicated. The specific row, which is activated by the pointer, is indicated by an arrow.

The pointer, initially reset to the first row, advances cyclically from row to row. In the first n clock cycles the first n data bits are written into the storage locations of the first column. Then in the clock cycle $n+1$ the pointer returns to the first row. The memory is organized in such a way that the data stored in the addressed row are read out first and then written in again, shifted to the right by one column. Thus data bit number 1 is shifted now to the second storage location and data bit number $n+1$ is written into the first storage location of the first row. Data in the other rows are processed in the same manner as the pointer advances from row to row. Each time that the pointer addresses a row, previously written data are shifted one column further to the right and a new data bit is written into the first storage location of that row. Thereby a data flow is established from the left side to the right side of the memory field. Each data bit stays in the row it was written in first and is shifted one column further to the right in every nth clock cycle. After a given number

$$D = (i \cdot n) + j, \qquad 0 \leqslant i \leqslant m; \qquad 0 \leqslant j \leqslant n-1 \qquad (1)$$

of clock cycles the left memory part, separated by the bold line in Fig. 5, is filled with data. The index i here gives the number of completely filled columns and the index j gives the number of used storage locations in column $i+1$.

Fig. 6 shows the status of this part of the memory after D clock cycles in detail. Circles indicate data written into the memory and numbers inside the circles give the sequence of writing. Data written in the first n clock cycles appear here at the right edge. The pointer has advanced to row number $j+1$. The first j rows have been addressed
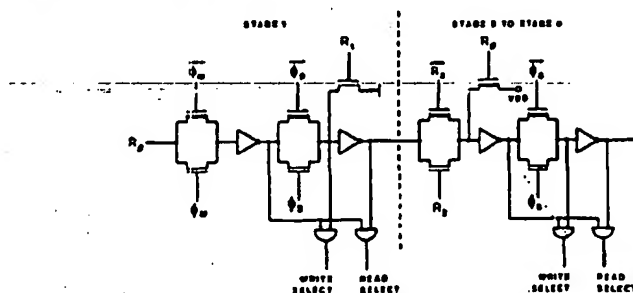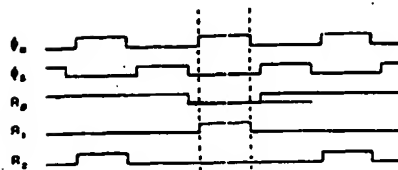
Fig. 3. Pointer circuitry.

Fig. 4. Pointer control signals.

one time more than the others, so that data have also advanced one column further. The step by one column, which appears in the filled memory portion between row j and row $j + 1$, has be to considered for the adjustable delay.

We now introduce an extraordinary reset of the pointer to the first row and continue cyclic operation for the next D clock cycles. Data will then cross the right edge of the memory part shown in Fig. 6, exactly in the same sequence as it was written into the memory and exactly D clock cycles after it was written into the memory. Thus for any given delay of data $D = i \cdot n + j$, only the read bit lines of column i and $i + 1$ are relevant. The correctly delayed signal appears on the read bit line of column $i + 1$ when the pointer addresses rows 1 to j. It switches from column $i + 1$ to the read bit line of column i when the pointer advances from row j to row $j + 1$, where it stays while the pointer addresses rows $j + 1$ to n. When the pointer returns to row 1 again, the correctly delayed signal switches back to the read bit line of column $i + 1$. In the special case $j = 0$, the correctly delayed data appear only on the read bit line of column i.

Therefore in order to realize a delay D, the two relevant neighboring columns i and $i + 1$ can be preselected statically. Dynamic switching, according to the pointer position, is only necessary between these two preselected col-

umns. The largest possible delay

$$D_{max} = n \cdot m \qquad (2)$$

is given by the total number of memory locations. Extension of this concept to a k-bit-wide data word is simply achieved by repeating the memory field of Fig. 5 k times. All of the k memory fields are accessed in parallel by just one pointer.

IV. EXPERIMENTAL CMOS CIRCUIT

A block diagram of our experimental CMOS circuit is shown in Fig. 7. This circuit serves as a delay line for a 4-bit-wide data word with a programmable delay between 1 and 4096 clock cycles. There are four main functional units, namely the memory field, the pointer, the multiplexer, and the control logic.

The memory field has 256 rows and 64 columns of three-transistor cells and is divided into four identical blocks of 256 rows and 16 columns, which are accessed in parallel by a common pointer. Each of these blocks is responsible for the delay of one bit of the data word. Incoming data bits are directly routed to their corresponding blocks in the memory field, where they are connected to the write bit lines of the first column. The outputs from all columns and also the incoming data bits are routed to the multiplexer, where static preselection of the two relevant columns and dynamic switching between these columns takes place. The multiplexer consists of four identical blocks, which are controlled by the same control signals. Each block is responsible for one bit of the data word. The multiplexer itself in combination with the input and output pad circuitry realizes a delay of one clock cycle. In case a delay of one clock cycle is selected, the memory field is not used and incoming data are directed via the multiplexer to the output pads.

The control logic generates the control signals for the multiplexer and the pointer. Inputs to the control logic are
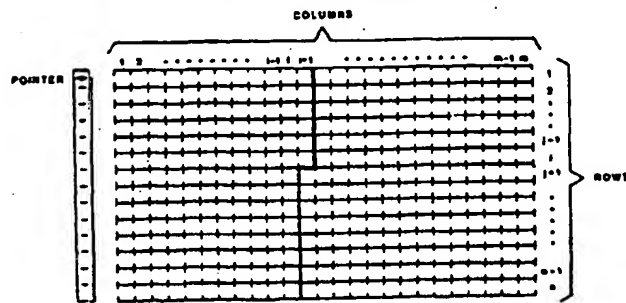
Fig. 5. Schematic representation of a memory field with n rows and m columns. The bold line indicates the edge for a delay $D = (i \cdot m) + j$.
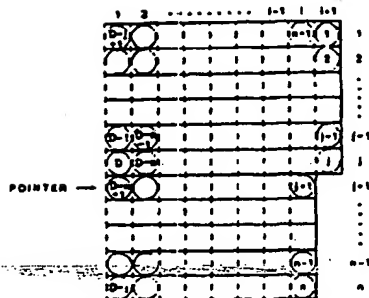

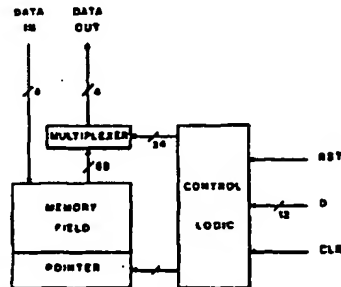Fig. 6. Status of the memory after D clock cycles.

Fig. 7. Block diagram of the architecture of the adjustable delay circuit. RST is an external reset, D is the delay programming word, and CLK is the external clock.
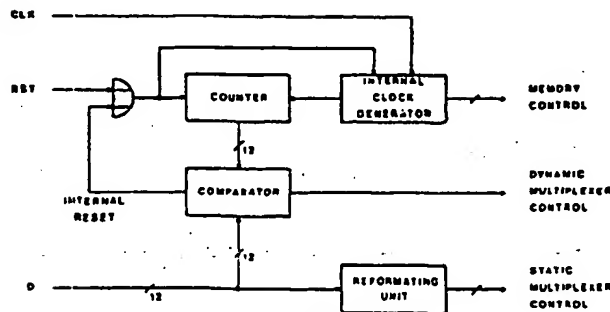

Fig. 8. Structure of the control logic.

an external reset signal $\overline{RST}$, a 12-bit delay programming word D, and the external clock CLK. Fig. 8 shows a block diagram of the structure of the control logic. The main components are an internal clock generator, a 12-bit counter, a 12-bit comparator, and a reformatting unit.

The clock generator generates from the external clock CLK a nonoverlapping two-phase internal clock and all control signals for the pointer including the pointer reset. The reset of the pointer to the first row of the memory field is synchronized with the reset of the counter to zero. Both resets are derived either from the external reset $\overline{RST}$ or an internal reset which is generated by the comparator, when the counter status is equal to the delay programming word D. The external reset serves the initialization purposes. The internal reset corresponds to the extraordinary reset needed by the delay-line algorithm as explained in the previous section.

The generation of the static and dynamic multiplexer control signals has to be adapted to the special organization of the memory field. In our experimental circuit there are 256 rows and 16 columns reserved for the delay of 1 bit of the data word. In this case (1) is written as $D = i \cdot 256 + j$ $(0 \leq i \leq 16, 0 \leq j \leq 255)$ and the important numbers i and j in our algorithm are given by the upper 4 bits and the lower 8 bits of the 12-bit delay programming word D, respectively.

The reformatting unit is responsible for generating the static multiplexer control signals by which the two relevant columns of the memory field are preselected for a given delay D. It decrements the delay programming word D to account for the delay of one clock cycle incorporated in the multiplexer and takes the upper 4 bits of the result to preselect column i of the memory field. Furthermore these upper 4 bits are incremented by 1 to preselect column $i + 1$.

The comparator in combination with the counter is responsible for generating the extraordinary pointer reset and the dynamic multiplexer control for switching between the two preselected columns i and $i + 1$. Since the counter and the pointer to the memory field are synchronized with every reset (internal or external), the counter status represents the position of the pointer. In particular, the lower 8 bits of the counter give the number of the row which is addressed by the pointer and the upper 4 bits of the counter give the number of circular passes of the pointer along the rows. Thus the extraordinary reset can be derived from the comparator when the counter status is equal to the external delay programming word D. Since column $i + 1$ has to be selected when the pointer is located between row 1 and j and column i has to be selected when the pointer is located between row $j + 1$ and row 256, the comparator switches to the selection of column i when the lower 8 bits of the counter are equal to the lower 8 bits of D and switches back to the selection of column $i + 1$ when the lower 8 bits of the counter are equal to 0.

A photomicrograph of the experimental circuit (Fig. 9) demonstrates the arrangement and area consumption of the different units. The memory field is split into two
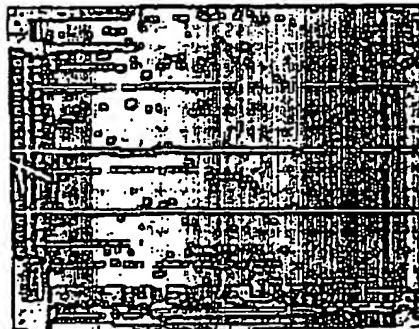

Fig. 9. Photomicrograph of the experimental CMOS chip.

halves by the pointer and occupies most of the chip area of 21.6 mm². On the other hand only about 10 percent of the chip area is used for the control logic and the multiplexer. The chip operates reliably in the frequency range from 3 kHz to 30 MHz. The basic characteristics of the chip and its technology are summarized in Table I.

V. CONCLUSION

A new concept for a digital delay line with a large arbitrarily adjustable length has been developed and verified by the design and fabrication of an experimental CMOS chip. The concept is based on a three-transistor cell memory with pointer access to the rows of the memory field. A special algorithm for pointer operation and a specific interconnection between columns of the memory field are utilized for the adjustable delay. The concept is characterized by the following advantages:

1) low power dissipation in comparison with shift registers or CCD solutions, because clocks do not have to be distributed to all storage units and only a small percentage of the data has to be shifted;

2) small area consumption and high operating speed, because of the three-transistor cell, which allows a READ and a WRITE operation in the same clock cycle;

3) small overhead for control circuitry, because of the pointer-controlled memory access and the built-in data flow in the memory field; and

4) inputs are directly wired to write bit lines in the memory field, so that no input address decoding is necessary.

ACKNOWLEDGMENT

REFERENCES

[1] M. Pelgrom and H. Terneer, "A 32 kbit variable length shift register for digital audio application," in *ESSCIRC Dig. Tech. Papers*, Sept. 1986, pp. 38–40.

[2] S. Matsumoto, I. Nakagawa, K. Kondo, and N. Kojima, "A video signal processing 2D on 2k×1f multifunctional memory," in *ISSCC Dig. Tech. Papers*, Feb. 1987, pp. 186–187.

[3] B. Zehner, H. J. Mattausch, F. Matthiesen, R. Tielert, and H.-J. Grallert, "A CMOS VLSI chip for filtering of TV pictures in two dimensions," *IEEE J. Solid-State Circuits*, vol. SC-21, pp. 797–802, Oct. 1986.

Fred Matthiesen was born in 1955 in Brake, Germany. He received the Dipl.-Ing. degree in electrical engineering in 1984 from the Technical University of Hannover, Germany.

Since 1984 he has been with the Research Laboratories of Siemens AG, Munich, Germany. Currently he participates in the integration of a video codec for ISDN. His special interests are in design for testability.

Jutta Härtl was born in Braunschweig, West Germany, in 1962. After graduating from high school in 1982 she got her professional education at the Siemens Technical School, Munich, West Germany.

Since 1984 she has been with the Research Laboratories of Siemens AG, Munich, where she is concerned with layout, testing, and simulation of integrated circuits.

Reinhard Tielert was born in Stolp, Germany, in 1945. He received the Dipl. Phys. and Dr. Ing. degrees from the Ruhr-Universität Bochum, Bochum, Germany, in 1972 and 1978, respectively. From 1973 to 1977 he was a Research and Teaching Assistant at the Ruhr-Universität Bochum, where he was involved in the development of gigabit electronics.

Since 1978 he has been with Siemens AG, Munich, Germany. There he worked first on process and device modeling of advanced MOS structures, including the development of a two-dimensional process simulator. Later he worked in the design of VLSI and communication circuits. He is currently leading a group of design engineers in the field of ULSI DRAM development.

Hans-Jürgen Mattausch was born in Hanau, Germany, in 1952. In 1977 he received the diploma in physics from the University of Dortmund, Germany. From 1979 to 1981 he was at the Max-Planck-Institut for Solid-State Research in Stuttgart, Germany. In 1981 he received a doctor degree in physics from the University of Stuttgart. His dissertation work was on the theoretical description of the electronic system of semiconductors.

He joined the Research Laboratories of Siemens AG, Munich, Germany, in 1981. He has participated in the design of fast SRAM's and generators for macrocells in semi-custom design systems, as well as in the integration of a video codec for ISDN.

Erwin P. Jacobs was born in Munich, Germany, on August 8, 1939. He received the Diploma and the Dr.rer.nat. degrees from the Technical University Munich in 1966 and 1971, respectively.

From 1970 to 1971 he worked as an Assistant Professor at the Institute of Physical Chemistry of the Technical University Munich. In 1972 he joined the Siemens Research Laboratories, Munich, Germany, and was involved in the physics of the Si surface and later in process development of nonvolatile MNOS memory devices. Since 1970 he has been engaged in advanced CMOS process design.

Dr. Jacobs is a member of the German Physical Society.

A First-In, First-Out Memory for Signal Processing Applications

NICK KANOPOULOS AND JILL J. HALLENBECK

Abstract —This paper describes the design and implementation of a First-In, First-Out (FIFO) memory for signal processing applications. The FIFO design allows concurrent input and output of data, both operations requiring one clock period each. The design is based on a 4 μ NMOS technology and operates with a 8-MHz clock. The length of the FIFO is programmable, resulting in minimum data ripple-through times for applications not requiring the full length of the memory. A built-in test scheme incorporated in the design makes the functional verification of the FIFO easy and it gives the memory some self-testing capabilities during operation. The memory size is 128 bytes and occupies 18.5 mm² silicon area. Memories of larger sizes are easily obtained by cascading FIFO chips.

I. INTRODUCTION

A First-In, First-Out (FIFO) memory is a read/write device that automatically keeps track of the order in which data is entered into the memory and reads the data out in the same order. The memory functions like a parallel-in parallel-out register whose length is always exactly equal to the number of words stored.

The most common application of a FIFO is as a buffer memory between two digital devices operating at different speeds. Even when two devices operate at the same data rate, it is not always possible for both to be operated synchronously. The FIFO provides the necessary data buffering to achieve synchronization, which is a requirement for many signal processing systems [1], [2]. It has been shown that a FIFO memory can be used for data shuffling during the computation of Fast Fourier Transforms (FFT) [3] and a FIFO can be utilized in array processor structures [4].

A problem associated with the use of FIFO memory is data latency, that is, the time for the data to ripple through the FIFO's stages. If the FIFO is interfaced with a processor, the processor's throughput can be lowered if the FIFO's data latency is higher than the data processing time. There are two cases where this situation can develop. The first is when the processor performs an operation and requests new input data faster than the time it takes for the data to ripple through one FIFO stage. The second and more common event, is when the processor performs iterative computations (i.e., FFT) on an input data record of N samples provided by a FIFO with R stages, where $R > N$.

The first case can only be accommodated by designing the FIFO with a data ripple rate faster than the computational rate of the processor. A solution to the problem imposed by the second case is offered here by designing a FIFO with programmable length. There are two implications due to this capability. First, the same memory can be used for signal processing applications with different input requirements without experiencing data latency; and second, if only part of the memory length is utilized, the power consumption is lower because the unused portion is not clocked and, therefore, does not dissipate dynamic power.

A very important aspect of the FIFO architecture is the design for testability scheme used to ease the functional testing burden

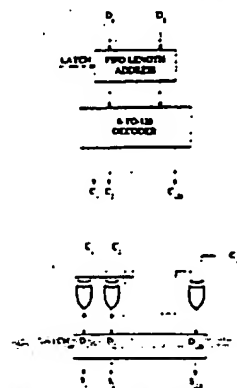Fig. 1. Logic design of the FIFO storage and control sections.

Fig. 2. The generation of the control signals C_i, S_i ($i = 1,128$).

at minimal cost, and at the same time used to give some self-testing capability to the FIFO. The signals generated during self-testing are available output signals and they can be used to achieve fault tolerance at the system level.

II. THE FIFO DESIGN

The logic design of the FIFO is shown in Fig. 1. All the signal terms used in the discussion concerning the FIFO design are illustrated in this figure. The C_i ($i = 1, \cdots, 128$) signals are provided by the decoder which is driven by the address specifying the desirable FIFO length for a given application. The decoder also drives the "hit-set" circuit that provides the S_i signals which only enable the operation of the FIFO control within the selected FIFO length. Fig. 2 illustrates the generation of these signals. The length of the FIFO is selected before data is stored in the FIFO. Data words are stored in 128 eight-bit registers connected so the output of one feeds the input of the next. The operation of the FIFO is performed by clocking each register independently so that data can be selectively shifted through the registers. Each register shifts independently based on the output of the cross-
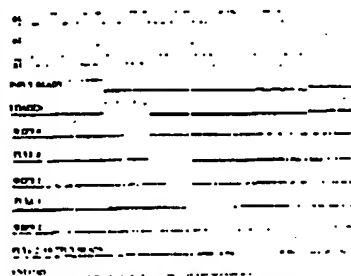
Fig. 3   Timing diagram for 3-FIFO stages

Fig. 4   Block diagram of cascaded FIFO's

coupled NOR gates associated with each register which determine whether or not that register contains valid data (Fig. 1).

Initially the FIFO is reset and there is no data stored in it. The FULL(j) ($j = 1, \cdots, 128$) bits are all reset to "0". When the LOADEN signal becomes "1", an 8-bit data word can be entered into the first register and the FULL-0 bit is set to "1", indicating that valid data is present in the first register. The FULL-1 bit of the second register is "0" and this causes it to continually monitor the FULL-0 bit of the first register looking for a "1". When the data is stored in the first register the control sees the "1" and generates a SHIFT-1 pulse which shifts the data from the first register into the second, sets the FULL-1 bit to "1" and resets the FULL-0 bit. The same process is repeated until the data arrives at the 128th register. At this point only FULL-128 (OUTPUT READY) is set to "1", the others having all been reset as the data was shifted into the next register.

As soon as the data moves from the first register to the second, the FULL-0 bit is reset to "0". A new data word can now be shifted into the first register. The new data shifts through the registers as long as their FULL bits are "0". Eventually the data reaches the register immediately preceding the one containing data, stores itself in that register since no further shifting is possible, and the process is repeated until all data words are entered.

When the UNLOAD line on the output goes "high", it causes the FULL-128 bit to reset indicating that the 128th register is empty. The next to the last word is shifted into the last register and the "0" on the FULL-128 line (OUTPUT READY) moves back toward the FULL-0 as the data words move down one register. This process can continue until all data has been shifted out of the FIFO. When the last word has been read, the FULL-128 (OUTPUT READY) bit remains "0" indicating that there is no data available at the output.

This scheme allows the reading and writing of data to occur completely independently. Data can be written into the FIFO as rapidly as one write per two cycles, after the LOADEN line goes "high". A timing sequence example for a three word FIFO is shown in Fig. 3.

The amount of time required for the first data word to ripple through the registers has been defined as data latency. The data latency can be computed by multiplying the clock period by the shift register stages. Since the length of the FIFO is programmable, the data latency is minimized for applications needing less than 128 input data samples.

In addition to the control signals required for the FIFO's operation, the SHIFT-0 (INPUT READY), FULL-128 (OUT-
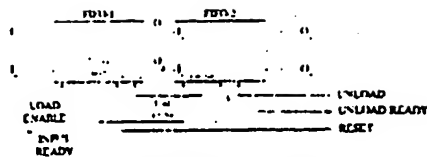
PUT READY) and CASCADE signals are also provided as outputs along with the 62 phase of the clock, thus making the synchronization and interfacing of multiple FIFO chips simple, therefore allowing the configuration of FIFO memories of any size. Fig. 4 illustrates how FIFO chips can be cascaded to obtain memories longer than 128 stages. In the case of two chips, FIFO-1 can be programmed to any length, while FIFO-2 is programmed to its maximum length. The CASCADE output of FIFO-1 and the INPUT READY output of FIFO-2 are not used.

III. FUNCTIONAL VERIFICATION AND SELF-TEST

Functional verification assures that data is stored properly and ripples properly through the FIFO stages. The functional verification is largely aided by the FIFO length programmability capability. This gives direct controllability to the inputs of the internal FIFO stages, a desirable testing feature. Using this capability during functional verification testing, the FIFO design can be checked stage by stage, starting with stage 128. In case a faulty stage is identified, active probing is used to define the fault at the transistor or interconnection level.

A self-test circuit is incorporated in the FIFO design to detect malfunction of the control section of the FIFO during the data ripple-through operation. This circuit monitors the consecutive storage of data in the storage unit. It is composed of a 128-bit shift-right/left register and a 128-bit comparator. The signals C, (Fig. 1) are used here to define the required length of the shift register and the signals S_i (Fig. 1) are used to enable the appropriate comparator stages.

When a word is loaded into the FIFO, the shift register shifts a "1" into the first bit location. This is a shift-right operation which is repeated every time a LOAD operation takes place. When data is stored in the FIFO, the FULL signal of each of the storage registers that contain data is "high" (Fig. 1), and the comparator compares the FULL-i lines of the FIFO with the Q lines of the 128-bit shift register (Fig. 5). If the two quantities are not equal, the comparator outputs a "1", indicating an error. In case of an error, the circuit is reset and the LOAD operation is repeated.

In the case of an UNLOAD operation, the shift register will shift-left a "0" and a comparison will be made. If 128 words are loaded into the FIFO before UNLOAD takes place, the shift register will contain 128 "1"s and the shift-left operation will not have any effect. To cover this special case, an extra bit is added to the 128-bit shift register (i.e., 129th bit) and this bit is always "0". If the LOAD and UNLOAD operations occur concurrently, the shift register does not shift left or right because the number of words remaining stored in the FIFO does not change. The logic design of the shift right/left register and its control circuitry as well as the design of the comparator are illustrated in Fig. 5. The VDD (i.e., power supply) connection to the shift register input provides the right shift of "1", while the VSS (i.e., ground) provides the left-shift of "0". The control signals for this circuit are the same signals used for the operation of the FIFO (Fig. 1).

Fig. 5. (a) Logic design of the shift register. (b) The comparator used in the location tree scheme.
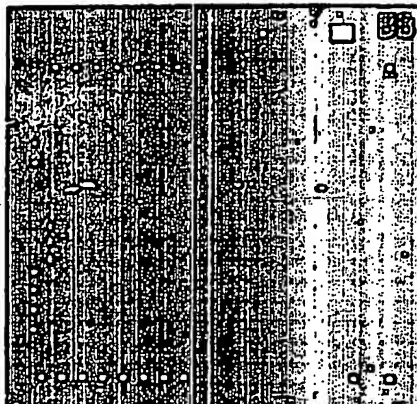

Fig. 6. Microphotograph of the FIFO chip.

IV. PERFORMANCE AND FEATURES

The FIFO memory chip can store up to 128 bytes. The chip occupies 18.5 mm² of silicon area, dissipates approximately 500 mW of power when its full length is utilized, and it is mounted in a standard 28-pin DIP (dual-inline package) package. The loading and unloading of the memory require one clock cycle for each operation and are synchronized with the clock which is provided by the 8-MHz on-board clock generator. Consecutive reads or writes can be performed as fast as one per two clock cycles. The two operations are independent of each other and thus can occur simultaneously. The memory outputs are tri-stated.

The FIFO design is based on 4 μ NMOS technology with rather conservative layout rules. Faster access times can be anticipated by implementing the same design using CMOS technology or using NMOS with smaller geometries.

The memory can interface directly with MOS or TTL technologies, having a driving capability of 25 pF. Fig. 6 illustrates a microphotograph of a prototype FIFO chip.

V. CONCLUSIONS

In many signal processing applications, there is a need for data buffering between machines operating asynchronously and for data shuffling in FFT-like operations. The FIFO design presented in this paper is an ideal solution to these problems. Its implementation utilizes a control scheme that makes the variation of its length possible upon user request, thus resulting in the minimum possible data latency for a given application. Its access time of 125 ns along with its capability for concurrent read and write operations and its self-test features, make this chip a prime candidate for a wide range of signal processing applications.

REFERENCES

[1] C. Seitz, "Self-Timed VLSI Systems," in *Proc. of Caltech Conf. on VLSI*, pp. 345-347, Jan. 1979.
[2] D. Hillman, "Intelligent buffer connects fast processors and slow peripherals," *Electronics*, Sept. 1980.
[3] N. Kanopoulos and P. Manson, "On the architecture of a programmable, high performance single-chip FFT processor," in *Proc. 10th European Conf. on Microprogramming EUROMICRO*, pp. 319-325, Aug. 1984.
[4] L. Johnsson et al., "Towards a formal treatment of VLSI arrays," in *Proc. Caltech Conf. on VLSI*, pp. 375-398, Jan. 1981.
[5] M. R. Buric and C. A. Mead, "Bit-serial inner product processors in VLSI," in *Proc. of Caltech Conf. on VLSI*, pp. 155-164, Jan. 1981.
[6] B. Gold and T. Bially, "Parallelism in FFT hardware," *IEEE Trans. Audio Electroacoust.*, vol. Au-21, pp. 5-16, Feb. 1973.

Model Reduction of Two-Dimensional Discrete Systems

E. I. JURY and K. PREMARATNE

Abstract —In this paper the one-dimensional (1-D) reduction method of Badreddin-Mansour is extended to two-dimensional (2-D) discrete systems. It is found by counterexample that contrary to the 1-D case, stability is not guaranteed for the reduced model, in general. However, stability is guaranteed for the reduced model if the original system is stable, in the following two cases:
(1) the original system is of the separable type; and/or
(2) the original system is of dimension one in each of the horizontally and vertically propagation sections, i.e., a 1½-1½ system.
Several examples are given to illustrate the reduction procedure, and its effect on stability.

I. INTRODUCTION

For realization, control, or computational purposes, it is usually desirable to be able to represent a high-order system by a lower order model. The fast development and use of small computers and processors in the design, analysis, and implementation of dynamic systems increases the importance of such

A 32-kbit Variable-Length Shift Register for Digital Audio Application

MARCEL J. M. PELGROM, MEMBER, IEEE, AND HENK A. H. TERMEER

Abstract — On this chip dynamic shift registers (DSR's) are combined with high-density serial–parallel–serial charge-coupled-device (SPS CCD) memory blocks in order to obtain a switchable chain of delay blocks with delay values that are powers of 2. The shift-register length can be adjusted from 17 to 32 767 clock periods. The trade-off between the delay implementations is presented. A detailed description of the SPS CCD is given. The chip has been realized in a 2.5-μm NMOS process with CCD option.

I. Introduction

THE successful introduction of digital audio equipment has stimulated the development of digital audio data processing with a view to implementing new features or enhancing the existing ones [1]. One of the most powerful improvements is the possibility of using (large) delays. Features like reverberation, compression, and scratch suppression can only be realized if the audio signal is delayed for several tens of milliseconds. This requirement corresponds (at 44-kHz sample rate, 24 bit/sample, stereo) to four to eight memory units each ranging from 1 up to 32 kbit. The first-in/first-out shift register is the most common organization of the memory units.

Fig. 1(a) shows the global setup of a digital audio signal processing unit with random access memories. The RAM's require an extensive control system in order to combine the READ–MODIFY–WRITE cycles of all the memory blocks at the required speed. The RAM controller is either a special-purpose IC or part of the digital audio processor chip. Its functions are address generation, which is limited to incrementing for a delay function, and input/output formatting. This configuration leads to pinning, packaging, part count, and PCB complexity problems.

In the setup of Fig. 1(b), serial memories of variable length are used. The total data exchange of the processor with the memory is the same, but the address generation is replaced by a simple control line which is used to organize the memory to the required format, thereby dispensing with the memory controller.

In this paper a serial memory is presented which can be used in the configuration of Fig. 1(b). The basic elements of this serial memory are dynamic shift registers (DSR's)
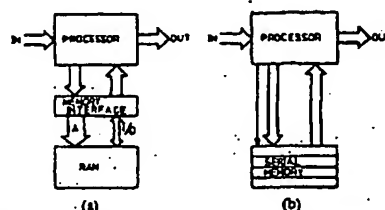
Fig. 1. Digital audio signal processing units. (a) Conventional setup, and (b) using serial memories.

and charge-coupled devices (CCD's). The requirements of this serial memory and its structure are presented in Section II. Section III deals with the delay implementation. The CCD configuration with clocking and input and output circuits is explained in Section IV. Finally a summary of the measurements is presented.

II. General Structure

As CCD technology has proved its value in video special-purpose memories [2] where the inherent serial character is an advantage, this technology is the obvious candidate for the creation of an audio serial memory. The basic requirements are that such a memory should:

a) be organized as a variable-length shift register up to 32 kbit;

b) be I/O and control compatible with the serial interface of the audio processor (the bits of each sample are put in series in order to make data transport more efficient);

c) have a shift speed from 0.5 up to 2.1 MHz (= 2×24 ×44 kHz);

d) have a 5-V power supply, and TTL compatible input and output.

The variable-length shift register has been realized as a combination of binary-weighted delay elements (Fig. 2). The input signal (in serial format) is clocked into the memory structure under control of the rising edge of the shift clock. It is transported by means of a chain of
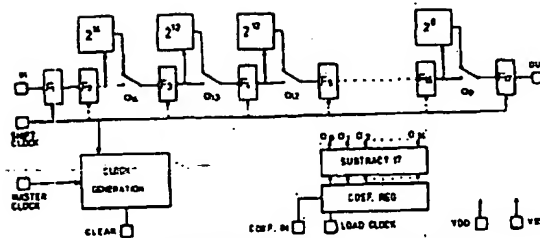
Fig. 2. Block diagram of the variable-length shift register.

switches and pipeline flip-flops F. The first and the last flip-flops are directly coupled to the external shift clock, and the intermediate 15 flip-flops are clocked by derived pulses. The inherent delay from input bondpad to output bondpad is therefore 17 shift clock periods.

The switches can alter the signal flow. If one or more switch is activated the signal has to pass through the associated delay section(s). The delays of the 15 sections have values that are powers of $2 : 1, 2, 4, \cdots, 16\ 384$. Any desired delay in the range from 17 to 32 767 shift clock periods between input and output can be realized. For the previously mentioned shift clock frequencies this delay corresponds to a maximum time delay of 16–64 ms. In fact a maximum delay of 32 784 clock pulses can be obtained, but the required number in the coefficient register is between 0 and 16. This situation is only used for testing purposes. The order of the delay sections in the chain is not important except for the first section. This section is always filled with the input data, and a reconfiguration of the switches will not affect its contents. The largest delay section is therefore first in the chain, and correctly delayed data are now available within half of the maximum delay time after reconfiguration.

The position of the switches is controlled via the coefficient register and a subtract-by-17 circuit. The latter circuit corrects the inherent delay of the 17 pipeline flip-flops. Usually the coefficient register is loaded through a relatively slow serial control bus.

The memory structures on the chip require a clock scheme that has eight periods within the shift clock period. As most digital audio systems provide a (synchronous) master clock that runs eight times as fast as the shift clock or faster, a start/stop clock generation circuit has been chosen for this chip. The positive shift clock edge starts the clock generation, which is stopped after eight master clock pulses. The resulting synchronization problem occurs only in the clock generation circuit, which decides whether to run or not. In the switch chain, where there are two interfaces between flip-flops clocked by the shift clock and the master clock, derived clock pulses with sufficient margins have been used. Alternative solutions like multivibrators or phase-locked loops either suffer from parameter variations or do not cope with irregularities in the shift clock.
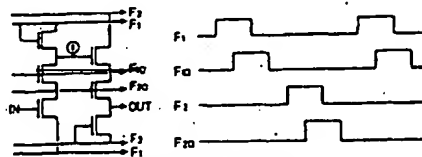

Fig. 3. The DSR cell.

III. DELAY SECTION IMPLEMENTATION

The 2.5-μm enhancement/depletion NMOS process allows the implementation of three types of shift registers: static flip-flops, DSR's, and CCD's. For the memory blocks on a 32-kbit chip, only the DSR cells and the CCD's have acceptable power and area consumption.

The DSR cell has been implemented in a six-transistor four-phase structure (Fig. 3). At the beginning of the shift cycle all clock lines are low and all nodes are isolated. Clock F_1 loads node 1 to a voltage one enhancement threshold less than the maximum clock voltage. F_{1a} will go high at, or after, the rising edge of F_1, but should remain high for a defined period after the falling edge of F_1. Now node 1 may discharge to the low level of F_1 if the input voltage is sufficiently high. After the first half of the clock cycle the shifted information (inverted) is stored on node 1. The same procedure is repeated with F_2 and F_{2a} to complete the bit shift. This shift-register cell has the advantages of a relatively small area (20×50-μm pitch) and no dc power consumption. The effective capacitive load of this cell is 40 fF for both F_1 and F_2, and 15 fF for F_{1a} and F_{2a}. At 1-MHz clock frequency one shift-register cell consumes 0.55 μA. The shift speed of the DSR is determined by the clock generation; charging the nodes via the diode-connected transistors requires (with 2.5-μm gate lengths) only a few nanoseconds. The lower frequency limit is determined by leakage current. The isolated diffusion areas pick up leakage current from the substrate or from generation processes at the locos edges where the diffusions touch the channel stop implantation. At leakage current levels of 100 nA/cm^2 (90°C), the lowest bit shift frequency would be 1 kHz, which is better than the lowest frequency for the CCD's.
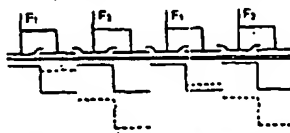
Fig. 4. The basic CCD shift-register structure. Solid lines indicate the surface potential at low clocks. Dotted lines are for active F_1 and charge level under F_1.

Fig. 5. Comparison of three delay implementations. (a) Area versus the block size. (b) Power dissipation trade-off at 1-MHz shift frequency.

The voltage restrictions on power supply and substrate voltage are determined by the effective node voltage. Although the influence of the substrate voltage on the threshold voltage (body effect) is small, a wide operating margin should be maintained. Apart from easy application, the operating margin is required because the internal substrate potential may vary significantly due to the large capacitive coupling and the considerable substrate resistance. The safety margin (defined as the additional node voltage before reading erroneous information can occur) for a discharged node 1 (passing on a ZERO) presents no problem, because it equals the transistor threshold voltage minus bootstrapping effects of the following stage. Basically the safety margin for passing on a ONE is determined by the quantity $V_{en} = V_{dd} - 2 \cdot V_t$, where it is assumed that the clock swings from ground to the power supply voltage V_{dd}, and V_t is the threshold voltage of the enhancement MOST (about 1 V). Some loss in V_{en} is suffered due to the capacitive feedthrough of the clock pulses (in a proper layout mainly determined by gate-diffusion overlaps) and to the residue charges after switching off the clocked transistors. An appropriate choice of geometry is necessary. The lower boundaries for proper operation will be determined by equating V_{en} to the required safety margin. The upper boundaries are determined by physical and technological limits, like breakdown, reliability, etc.

The third memory technology (CCD) makes use of an n-channel surface CCD technology. The basic CCD cell is formed by two gate layers with a built-in potential barrier (Fig. 4). The first polysilicon storage gate is identical with the standard enhancement MOST. The barrier gate (with threshold voltage V_{fl} of about 4 V) is created by an implantation under the second gate (aluminum) which has a gate oxide that is two and a half times the gate oxide of the first layer. The CCD is clocked with a two-phase nonoverlapping clock, identical with the F_1 and F_2 clocks of the DSR. The use of these clocks allows easy adaptation of the input and output circuits of the CCD to the DSR timing, thus simplifying the logic design. The pitch of the gates is 8.5 μm. In a single-line shift register (with two-phase clocking) two stages are needed per bit. Together with interconnection lines for the gates, one bit in a single-line CCD would require 17×30 μm^2, which is only half of the DSR cell.

In the serial-parallel-serial (SPS) structure, the well-known interlacing and ripple clock techniques (e.g., [3]) can be used, which reduce the area and power consumption considerably with respect to the single-line CCD.
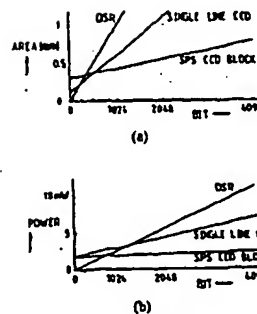
Some area overhead has to be taken into account for more complex wiring, input and output shift registers, circuits, etc. The maximum SPS block size in this design is 4096 bits. See Section IV for a detailed description.

Fig. 5(a) compares the areas of the DSR, the single-line CCD, and the SPS CCD block (with the above-mentioned structure) as they are in the final chip as a function of the memory block size.

The power consumption of the SPS CCD block is composed of the dynamic dissipations of the fast input and output shift registers, and the slow parallel registers and the dc dissipation of the input and output circuits. Based on a 4-kbit block the dc current is 360 μA, which is incremented by 0.02 μA/bit at a bit shift frequency of 1 MHz. A comparison with the single-line CCD and the DSR is made in Fig. 5(b).

The area advantage for the single-line CCD exists for only two of the required delay sections; it has no region with a power consumption advantage. This structure has been rejected for the chip design. The crossover point between SPS CCD blocks and DSR is at 400 bits for the area comparison and at 680 bits for the power dissipation comparison. This limit and the fact that a 512-bit SPS CCD block turned out to break the regularity of the layout led to the decision to implement the delay sections from 1 to 512 bits in shift-register technology and to design larger delays with 1-, 2-, and 4-kbit SPS CCD blocks.

The power supply restrictions in the CCD structure of Fig. 4 require the surface potential of the second-layer gate at V_{dd} to exceed the surface potential of the first-layer gate at ground potential. The first-order approximation for the lower boundary of the power supply shows that $V_{dd} - V_{fl} + V_t$ must be positive. As the second-layer threshold has a body-effect factor that is roughly 2.5 times the first-layer body-effect factor (mainly due to the gate-oxide difference); the variation of $V_{dd} - V_{fl} + V_t$ as a function of the substrate voltage is in the same range as the variation of V_{en} in the DSR. The restrictions on V_{dd} and on the substrate voltage for the basic CCD cell and the DSR cell warrant the expectation that their combination in one chip

will allow sufficient margin for supply variations. More-over, the clock generation can be shared and the data interface between the two shift-register types presents no problems. The DSR and CCD do not match with respect to process variations. The second gate level in the CCD is formed by means of an additional implantation and oxida-tion: their variations will be independent of the first gate level and the enhancement transistor.

IV. CCD Memory Implementation

In the general survey of the audio serial memory some considerations have been presented with respect to the design of the CCD blocks. Further elaboration includes three distinct parts of the design: the SPS structure, the clock generation, and the input and output circuits.

A. Serial–Parallel–Serial Structure

The gate structure of the SPS memory block is given in Fig. 6. At the top and bottom the serial input and output registers are shown. Each storage well in the serial input register is connected via a parallel channel to the corre-sponding storage well in the output register. The serial and parallel channel gates have been implemented with the barrier gates connected to the storage gates and are driven with a drop clock system [3]. Alternatives in a two-gate-layer technology are push clocks and four-phase clocking. In both cases the charge has to be stored under gates with a low-impedance connection to the positive supply voltage in order to avoid charge spillover due to capacitive cou-pling to the adjacent gates (especially in the ripple clock system). In an NMOS technology a low-impedance con-nection to the positive power supply is hard to realize: bootstrapping wide enhancement devices requires large capacitors and low leakage depletion devices, whereas a solution with large depletion loads increases the power consumption. The drop clock approach offers less charge storage, but can be made less vulnerable to capacitive coupling. Moreover the charge storage capability does not depend on the positive power supply.

The shift operation in the serial register allows only half of the wells to be filled. The parallel registers are therefore loaded by shifting in two lines of data which are inter-laced: the first data line is transferred to the parallel channels if all odd-numbered storage wells are filled, and the second data line is transferred if all even-numbered wells are filled. A dump drain on the serial output channel sinks the leakage current that is collected after parallel transfer.

The transport mechanism in the parallel channels is a ripple clock system [3]. The decision to use a four-phase ripple clock was made after considering the consequences for area and power consumption. Increasing the number of clock phases reduces the SPS block area, but requires more wiring and drivers. The power consumption is determined by CV^2 for a low number of clock phases and by the driver dissipation for a high number of clock phases. Fig. 7
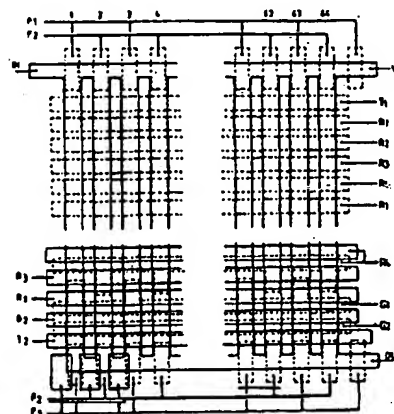

Fig. 6. The SPS structure. The active area and the polysilicon gates (dashed lines) have been indicated. In the lower part of the parallel channels the aluminum gates have been indicated as well (thin lines). The signal names correspond to Fig. 9.
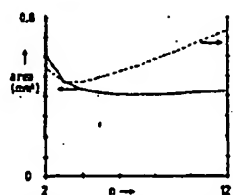

Fig. 7. Power dissipation and area of the ripple system versus number of ripple phases. Data are given for a 4-kbit block and for an input shift frequency of 1 MHz.

shows the power consumption and the area of the ripple gates and the ripple circuits and wiring per 4-kbit block using typical process parameters as a function of the number of ripple clock phases. The effective area is 103 μm^2/bit for a four-phase ripple.

Just before the output serial register the charge packets in the parallel channels are deinterlaced: a comb structure, controlled by two second-layer gates, allows the packets in the odd-numbered channels to pass to the output register and delays the packets in the even-numbered channels until the output register is free again. This deinterlacing structure, which resembles an image sensor structure [4], separates the deinterlacing and the parallel-to-serial trans-fer functions; the transfer pulses do not have to meet any special requirements. No intermediate dc gates (e.g., [5]) are needed. The deinterlacing pulses are taken from the ripple clock scheme. An alternative solution to this inter-lacing structure is charge multiplexing in the parallel part [4]: in the transfer pulse clocking generator two NOT-AND-OR gates are saved because serial-parallel and parallel-serial transfer occurs only from and to the odd or
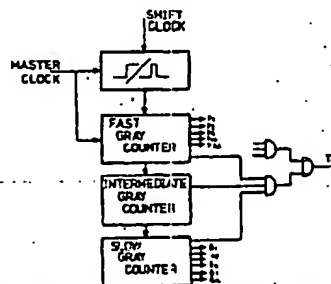
Fig. 8. Block diagram of the clock generation. The logic for defining a transfer pulse is indicated.

Fig. 9. The ripple, deinterlacing, and transfer clock pulses. The enlarged timing shows the transfer pulses at odd- (o) and even- (e) numbered input lines.

the even serial channel wells. However, the multiplex/demultiplex structures require more space in the SPS blocks and additional clock lines are needed.

The size of the largest CCD block (4096 bit) is determined by the lowest bit shift frequency (0.5 MHz) and the leakage current at the highest operating temperature. Correct charge detection in the SPS output circuit can only occur if the collected leakage current is not disturbing the charge representing the information. An ill-situated well (in the corners near the output) may collect up to eight times the charge of a well in the middle of the matrix. A leakage current level of 100 nA/cm² will introduce charge in a corner well corresponding to 0.1 V on the input gate if the charge is refreshed every 8 ms.

A convenient number of parallel channels is 64, because the 4-kbit block is square (minimum wiring), the clock counters are simple, and the 1- and 2-kbit blocks can be made by leaving out a number of ripple clock stages. In the 4-kbit block 21 ripple clock sections store 63 lines of data. One line of data is involved in the input, output, and interlacing sections. The 20th ripple clock section from the top has a wider gate pitch in order to allow the aluminum gates to be used as parallel connections.

B. Clock Generation

The clock generation has been split into four parts: a start circuit and three counters. The start circuit forms the digital derivative of the shift clock. The fast Gray counter generates, on a start input pulse within a sequence of eight master pulses, the serial CCD shift pulses which are also used for the DSR. If no new start pulse is available the counter goes into a waiting state. One pulse of the fast counter drives the intermediate counter which divides by eight. The slow counter finally generates the ripple clock pulses. All three counters are 3-bit synchronous counters, which is a compromise between the propagation delay of an asynchronous counter and the area of a full 9-bit synchronous counter. The Gray-code counters allow derived pulses that do not suffer from glitches, which would cause unwanted transfers in the CCD.
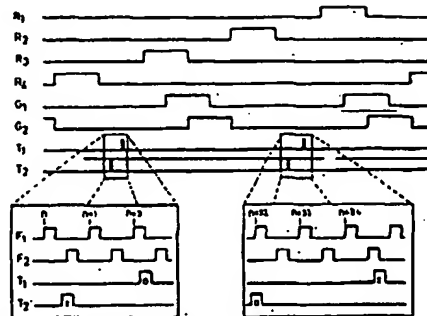
The generation of the transfer pulses requires the combination of pulses of all three counters. As shown in Fig. 8, the coupling pulses that connect the counters have been chosen in such a way that the transfer pulses have sufficient margins with respect to their gating pulses from the slower counters. In Fig. 9 the pulse diagram is shown with details of the transfer pulses for odd and even counts.

It has been noted before that a main problem in CCD's with on-chip clock generation is the clock feedthrough [6]. The unwanted but unavoidable gate overlap in the ripple clock section will couple clock changes on phase R_n to R_{n-1} and R_{n+1}. The problem is illustrated in Fig. 10(a) and (b), where it can be seen that feed forward of charge can occur. For rising edges there is a simple remedy, which is to change the ratio of the pull-up and pull-down transistors in the ripple clock drivers. However, for the falling edge the pull-down transistor also serves as clamp transistor; enlarging the W/L ratio only speeds up the feedthrough; the top value remains unchanged. An equivalent circuit for analyzing this problem is shown in Fig. 10(c). It is assumed that the driver impedance is constant and that the internal driver time constants are small with respect to the clock-line charging. The resistance of the active driver differs from the resistance of the clamping drivers. Laplace analysis on the diagram of Fig. 10(c) shows that the expression for the maximum of the clock feedthrough from a switching gate on an adjacent gate is

$$\Delta V = \frac{2V_{dd}}{A+B} \cdot \left(\frac{A-B}{A+B}\right)^{(A-B)/2B}$$

with

$$A = \left(\frac{R_1}{R_2}+1\right) \cdot \left(\frac{C_2}{C_0}+2\right)$$

and

$$B = \sqrt{\left(\frac{R_1}{R_2}-1\right)^2 \cdot \left(\frac{C_2}{C_0}+2\right)^2 + 8\left(\frac{R_1}{R_2}+1\right)}.$$
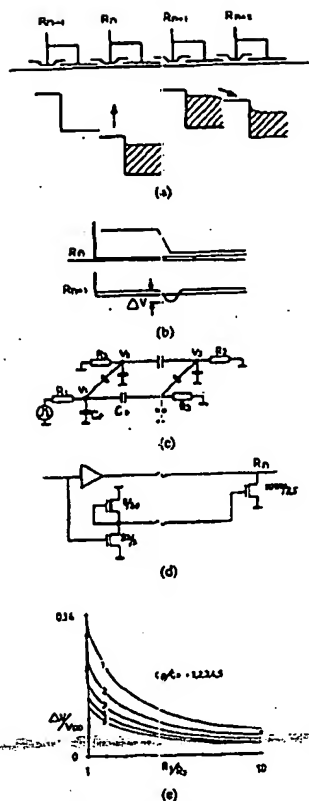
(a)

(b)

(c)

(d)

(e)

Fig. 10. The ripple clock feedthrough. In (a) and (b) the problem is depicted, (c) shows an equivalent circuit for the four-phase ripple clock, in (d) the solution is shown, and (e) gives the top value of the clock feedthrough as a function of resistor and capacitor ratios.

In Fig. 10(e) this formula has been plotted as a function of the resistor and capacitor ratios. If no precautions are taken the resistor ratio equals 1 and the typical capacitor ratio is around 2. In this chip additional clamp transistors have been placed as close as possible to the SPS blocks (Fig. 10(d)). They enlarge the resistor ratio to 10. The significant reduction which is predicted has been experimentally verified (see crosses in Fig. 10(e)) by laser cutting the clamp-transistor connections.

C. Input and Output Circuits for the CCD

The charge levels that correspond to both logic levels are determined by the input circuit. The low charge level corresponds in a surface channel device to a small charge
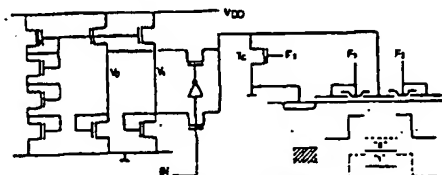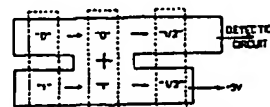

Fig. 11. The input circuit of the CCD.

Fig. 12. The reference CCD.

packet to provide a "fat zero" charge. The high charge level is ideally a safety margin beneath a full well (to allow some leakage current). This means that the high charge level must be derived from the second-level threshold V_{t2}. This threshold has a high substrate doping dependence, which leads to unwanted charge-packet modulation in the event of substrate potential variations. Therefore both charge levels have been derived from the first-layer threshold.

Fig. 11 shows the input circuit. The ZERO and ONE voltages are fractions of V_t determined by the W/L ratio of the transistors. The effective ZERO voltage is $0.2V_t$ and the ONE voltage is $0.9V_t$. Two pass transistors select the level that is applied to the input gate. The charge transfer from the input gate into the channel is facilitated by the chop transistor T. The input gate is pulled back to ground if charge transfer is required. (An alternative solution is bootstrapping the first transport gate.) Note that clock F_1 isolates the input section to prevent back flow of charge. If necessary the F_1 sample gate can be bootstrapped to compensate V_{t2}. The F_1 sample gate in the input circuit determines the overall voltage characteristic because $V_{dd} - V_{t1}$ must be positive and V_{t2} must exceed $2V_t$, which yields $4.5 > V_{t2} > 3.0$ V. The output circuit consists of the charge reference generation and the detection circuit. Temporal supply variations have little influence on the input circuit; there is no need to generate the reference charge and the signal charge at the same time and delay the reference charge packet for the same time period as the signal charge packet. A small dual-input CCD line (Fig. 12) accompanies each CCD SPS block. At the two inputs a ZERO and a ONE charge level are generated, then added and split into equal parts in the charge domain. One reference charge packet is used for detection, the other is destroyed.

The detection circuit consists of the CCD output, a simple amplifier, and the latch (Fig. 13). The CCD sense node is reset at the beginning of a detection cycle to a voltage just under $V_{dd} - V_t$. The reset transistor can be driven by a V_{dd} clock pulse, and need not be bootstrapped.
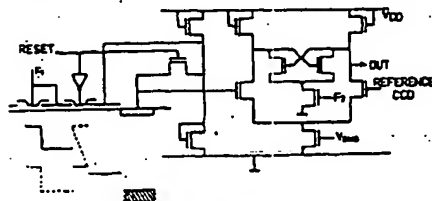
Fig. 13. The detection circuit.

TABLE I

power supply	5 volt @ 22 mA
	-5 volt @ 10 μA
I/O levels	TTL compatible
chip area	3.35×3.55 mm²
process	2.5 μm NMOS/CCD
max. shift freq.	4 MHz
min. shift freq.	0.8 MHz @ 100°Camb.
	0.1 MHz @ 70°C amb.

Fig. 15. Schmoo plot of the 32-kbit variable-length shift register.

V. RESULTS

The circuit has been successfully processed, and some results are summarized in Table I. Fig. 14 shows the chip and Fig. 15 gives a Schmoo plot at room temperature and 2.5-MHz shift frequency. The dynamic shift register operates above 4 V. The slope in the curve is the substrate influence on the two thresholds. The CCD sections determine the operating margins of the entire chip: at high V_{dd} and low substrate potentials the charge wells are too small for the charge packets, and at low V_{dd} the V_{t1} threshold curve is visible. At 70°C the V_{t2} curve is slightly lower (0.1 V), which indicates a threshold temperature coefficient of −2 mV/°C.

The chip has been designed with a bondpad layout and power supply wiring that allows the chip to be quadrupled to a 128-kbit device without significant changes.
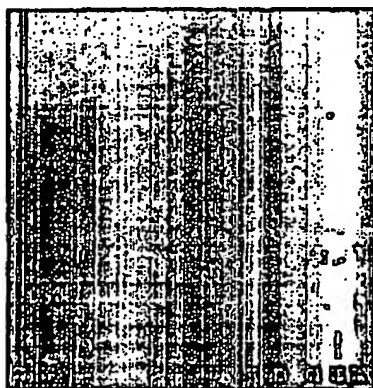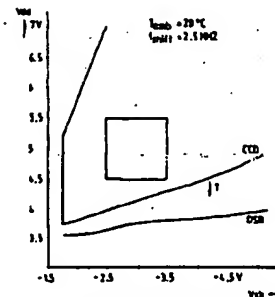
Fig. 14. Chip photograph of the 32-kbit variable-length shift register.

(At unfavorable power and processing conditions bootstrapping close to the unprotected sense node can cause spurious charge injection by parasitic inversion, hot electrons, or charge pumping.) The last gate before the sense node is at the sense-node reset voltage, leaving about one threshold-voltage swing for the sense node. The maximum voltage that a charge packet can induce is therefore limited to V_t. The purpose of the differential stage is to shift the signal back to V_{dd}; the amplification must be limited in order to avoid overdriving the circuit. The loads of the differential pair serve too as loads for the latch. The charge is put on the sense nodes at the falling edge of F_1, and the latch is activated by F_2.

REFERENCES

[1] E. H. J. Persoon and C. Vanderbulcke, "Digital audio: Examples of the application of the integrated signal processor ASP," Philips Tech. Rev., vol. 42, pp. 201–216, Apr. 1986.
[2] M. J. M. Pelgrom et al., "A digital field memory for television receivers," IEEE Trans. Consumer Electron., vol. CB-29, pp. 242–250, 1983.
[3] W. F. Kosonocky and K. H. Zaininger, "Applications of CCDs to memories," in Charge-Coupled Devices and Systems, M. J. Howes and D. V. Morgan, Eds. Chichester, U.K.: Wiley, 1979.
[4] R. L. Angle, J. E. Carnes, W. F. Kosonocky, and D. J. Sauer, "Techniques for the design of high-density high-speed TDI CCD image sensors," in Proc. Int. Conf. Application of CCDs (Edinburgh, U.K.), 1978, pp. 1/1–12, fig. 9.
[5] R. C. Varshney and K. Venkatarwaran, "A block organized 64-kbit CCD memory," IEEE J. Solid-State Circuits, vol. SC-13, pp. 681–687, Oct. 1978.
[6] Y. Gamou et al., "All TTL compatible CCD memory with CCD clock generator," IEEE J. Solid-State Circuits, vol. SC-15, pp. 881–886, Oct. 1980.

Marcel J. M. Pelgrom (M'83) was born in Zevenaar, The Netherlands, on September 17, 1952. He received the Ing. degree from H. T. S., Arnhem, The Netherlands, in 1974, and the M.Sc. degree in electrical engineering from Twente University of Technology, Enschede, The Netherlands, in 1979.

In 1979 he joined Philips Research Laboratories, Eindhoven, The Netherlands, where he was involved in the design of CCD memories. His current interests are in the field of analog integrated circuits and CCD's.

Henk A. H. Termeer was born in Sint Oedenrode, The Netherlands, on February 29, 1948. He received the degree in electrical engineering from Eindhoven Technical College, Eindhoven, The Netherlands, in 1967.

In 1970 he joined Philips Research Laboratories, Eindhoven, The Netherlands, where he is engaged in electrical engineering and IC design of MOS and bipolar circuits in the Consumer Electronics Group.

D

PRECISION TIME TRANSFER IN TRANSPORT NETWORKS USING
TAL CROSSCONNECT SYSTEMS

W.D. Grover and T.E. Moore

Alb. (elecommunications Research Centre (ATRC)
4245 97 Street, Edmonton, Alberta, Canada, T6E 5Y4   (403) 461-3830

ABSTRACT: We describe a technique for precise synchronization of the time-of-day clocks in networks of Digital Cross-Connect Systems (DCS) in order to enhance the performance of reconfigurable transport networks. The method is specifically devised to exploit the fine time resolution of the carrier signals to which a DCS has direct access. The resulting scheme is a master-slave, multi-site, implicitly-delay compensated, non-hierarchical time-transfer method with a theoretical precision of one bit time at the carrier rate. In practice, precision is limited by transmission span delay asymmetries but residual time errors of under one microsecond are predicted. The method is intended for n/a space-switching DCSs but other DCS properties can be accommodated. Requirements for DCS equipment design are given including a generic circuit module for DCS hardware support of the time transfer function. The proposed method applies to DS-3 or SONET networks and requires no change in the standards. Measurement or improved estimation of characteristic span delay asymmetries is recommended to refine the preliminary performance estimate.

I. INTRODUCTION

1.1 Objectives and Motivation

Todays synchronous network is at 8 kHz frequency-locked network but not a time synchronous network [1]. Nonetheless network-wide time-of-day clock synchronism is an important feature for the coordination and management of future DCS-based dynamic transport networks. The goal of this work is a practical method to achieve sub-microsecond absolute time synchronization of the real-time clocks in a network of DCS machines. The work was stimulated by recent ECSA T1X1.4 committee interest in time synchronism for SONET [2] for precise coordination of network operations. A previous proposal by Ellson [3] is a non delay-compensated broadcast time distribution scheme [3] which would have residual time errors of 10 to 20 msec. depending on propagation delays.

We propose a slightly more complex scheme for DCS-based networks which achieves master-slave, multi-site, implicitly-delay compensated, non-hierarchical time dissemination using a technique specifically devised to exploit the fine time-resolution of the carrier signals to which a DCS has direct access. Residual network timing errors under 1 μsec are predicted in practice and timing errors as low as 20 nsec are achievable in the limit.

Sub μsec time-of-day synchronization would ensure the co-ordination of simultaneous multi-site operations that reroute live traffic without call-dropping and would minimize disruption to data services during dynamic reconfigurations. Precision time information can also improve the capability of alarm-correlation diagnostics by improving event time-stamping accuracy. With sub-μsec nation-wide time-transfer, the telecom network could even become a backbone for new commercial time services in the navigation and broadcast industries. With 100 nsec precision, three cooperating metropolitan sites can in principle triangulate the position of a mobile unit within tens of meters. This may have application in future cellular radio systems.

1.2 Classification of Time-Transfer Methods

To relate our scheme to previously reported methods for time-transfer, we use a classification developed from consideration of references [4-14] in which systems are considered by these attributes:

a) Implicit or Explicit delay compensation If a time-transfer scheme uses one operation to measure propagation delay(s) and a separate operation to transmit delay-compensated time, we consider it an explicitly delay compensating type. In implicitly compensated schemes the source of reference time information does not modify its transmission and no process measures absolute delays to distribute compensation information to the time receiving sites.

b) Number of sites simultaneously synchronized: A scheme is either multi-node or two-node depending on the basic synchronizing process. Schemes which achieve network synchronization by a consecutive series of two-node steps are considered two-node schemes.

c) Mutual or Master/Slave synchronization: A synchronization scheme may result in alignment of dependent sites to an unmoving reference (master-slave), or convergence to a mutually derived arbitrary convention (mutual synchronization).

d) Hierarchical or non-hierarchical: We distinguish between a multi-site synchronization scheme that depends on a hierarchy of repeated time transfer operations and one which requires only one level of time-transfer operation.

e) Averaging or Direct operation: We consider whether a scheme accomplishes time-transfer with a single operation or if time-transfer requires averaging or convergence. A direct method may be repeated as desired for possible averaging advantages but in such a case the intrinsic mechanism is still classed as direct.

1.3 Present Methods for Time-Transfer

Lindsay gives a classification of methods based on type of delay compensation [4]. Two dominant implicitly compensated methods which he reports [6,7] are both mutual synchronization schemes. Other multi-node schemes [8,9,10] are also mutual synchronization methods. Apparently absent from the literature on multi-node schemes is a method which does not rely on mutual synchronization or a series of two-node synchronization operations to achieve multi-node time-transfer. This is a key property of the proposed method.

Precision two-site time-transfer has been reported using geosynchronous satellites [11,12]. Time-transfer via satellite can achieve precision from 5 to 100 nsec but can be equipment and signal processing intensive. Satellite time-transfer and the donor-homogeneous radio scheme reported by Ellingson [5] have a similarity to the scheme shown here because they employ round-trip propagation halving, but both of the former are inherently two-site schemes. The method in [5] also obtains its delay-compensation information from different timing measurements than in the scheme here. Other schemes differ from the proposed method by using explicit delay measurement and a series of two-node operations to achieve multi-node synchronization. These include a two-point master slave round-trip delay halving scheme [13], a similar multi-site explicit delay measuring scheme [14] and a scheme [15] using a hierarchy of two-point master-slave explicitly delay-compensated stages.

Each of these schemes has drawbacks for use in the transport network application. Satellite time-transfer requires earth stations at each node. The mutual synchronization schemes require continuous operation and/or long time constant and are complex in behavior with respect to locking to an external reference, the addition of new sites, and overall dynamics and stability. In principle, some hierarchical and/or explicit delay measuring schemes could suit the DCS application but they require more sub-operations than the following method which requires no equipment or facilities other than the transport network in which the DCS already resides and a special DCS circuit card. In relation to previously reported methods, the proposed scheme is an implicitly-delay compensated, multi-site, master-slave, non-hierarchical, direct transfer method.

47.2.1.

01/10/2001 WED 13:52 [TX/RX NO 8821]

PATENTEC FILES

MISSING PAGE(S) FROM THE U.S. PATENT OFFICE OFFICIAL FILE WRAPPER

Page 1545

III. IMPLEMENTATION

1 DCS Time-Transfer Support Hardware

In Fig 2 a time-transfer card was shown which would interface to a DCS core using two normal port appearances and provides the hardware and control functionality necessary to support operation of a node as a *trigger*, *loop*, or *intermediate* node. The internal architecture of such a card is shown in Fig 3. Configuration and control of the sub-circuits in Fig 3 (i.e. execution of the protocol which achieves time-transfer) is performed by the Finite State Machine (FSM) controller shown in Fig 4. The hardware in Fig 3 receives coarse time data such as year, month, day, hour, and minute from the DCS Operating System (OS). Only seconds and sub-second time data need be subject to the precision time-transfer mechanism. The support hardware has two normal bidirectional port appearances to the DCS core. These are *[t1port (in), blue/p (out)]* and *[t2port (in), t2port (out)]*.

In Fig 3, the m_1 and m_2 *detectors* decode the m-code to start and stop the $C(t)$ counter at an intermediate node i. If a DCS is the *trigger* node, the 2:1 multiplexer associated with the m_1 detector selects the m-code insertion command to start the $C(t)$ counter in that particular case. The $R(\cdot)$ latch captures the sampled *loopnode* clock data following the m-code on t2 path. The *adjust* block halves the $C(t)$ interval count and adds β (Appendix A) to compensate for the non-zero arrival time duration of the $R(\cdot)$ data and any other processing delays. The *Own R(t) Sender* register and associated 2:1 multiplexer permits operation as the *loopnode*. In this role, the m_1 detector output strobes the local $R(t)$ register into the *own R(t) Sender* register for immediate serial transmission following m-code on the loopback path.

1.1 Requirements on DCS Specifications

The proposed method for time-transfer requires *no changes to the DS-3 or SONET signal standards* because it works over out-of-service transport entities and the supplies its own working signal. It does however, require the following features of the host DCS:

Figure 3: Structure of High Precision Time-Transfer Card

4. This should be a common feature in DCS machines because there are important other uses for this ability. Examples are bridge-to-east-port, bridge-for-loopback, bridge-to-west-port and various replication uses.

5. Our studies have identified a further means that could compensate for arbitrary DCS delay asymmetries using progressive time-transfer and deduction procedures but that alternative is considerably more complex.

N/N Crossconnection The method preferably assumes a true space-switching crossconnect functionality where the full time-resolution of the transmission carrier signals is available to the time-transfer process. DS-3/2 and SONET crossconnects with STS-1 or STS-3 granularity are envisaged as the preferred machines to support time-transfer but one can adapt the concept to N/X DCS functionality, with some loss of performance, through two approaches:

a) migrate the method to a lower transmission level. In a 3/1 DCS for example, apply the method as described but at the DS-1 level within a DS-3 where the given DCS provides an apparent space-switching function.

b) adapt the method to work via overhead fields. This might arise in an STS-1 Virtual Tributary (VT) crossconnect which switches VTs only and always reconstructs new outgoing STS-1 signals. In this case true carrier-rate space-switching is not provided. Time-transfer could be adapted to work in this environment through reservation of appropriate path-level [2] overhead in the STS-1 signal format.

Bridging Connections The host DCS should preferably support bridging connections for convenient setup at intermediate nodes. However, if a DCS does not provide bridging, the method can be adapted to work in a manner in where the time-transfer card is placed in series in the t1 path and t2 path, providing its own internal bridged signal access.

Figure 4: Finite State Machine for Control of Time-Transfer Hardware

47.2.3.

DCS Path Delay. There is no requirement on absolute path delay for the host DCS machine but it is important that there is *nominally constant delay* for any one-way path through the DCS. If a DCS has several nominal delays, due to internal switch core properties, there is no difficulty if the delays are known to the OS so that equal delay paths through the DCS matrix can be selected, or (by extending the method slightly) so that the time-transfer card can compensate for any known DCS path delay mismatch.

IV. PERFORMANCE

4.1 Performance Estimate

Appendix A shows that the precision attainable with this method is limited by the *net delay asymmetry* over the time-transfer path between any particular code and the loop node. The residual timing accuracy is therefore determined by delay asymmetries in DCS machines, cross-office cabling, and transmission systems. Equipment delay asymmetries have not traditionally been measured or specified in transmission products and are difficult to estimate in a general way. Quantification of practical field performance is therefore difficult without field measurements to estimate the network-wide variance of delay symmetry in inter-DCS transmission paths. Jitter and wander also contribute a time-varying component to path delay asymmetries. However, jitter levels in DS-3 networks are generally well below 1 Unit Interval (UI) pk-pk. As a plausible worst case estimate, assume DS-3 operation and allow 1 UI rms jitter on each span, in each direction in the time-transfer chain. Also allow 2 UI rms delay asymmetry in every DCS, and 8 UI rms delay asymmetry in each transmission span. If the delay asymmetries are uncorrelated, then the maximum variance of timing uncertainty in a time-transfer path of n nodes will be (from Appendix A):

$$\sigma^2 = [(n-1)/4](2^2 1 + 2^2 2 + 8^2 8) \; UI^2.$$
$$\sigma = 8.38 \, sqrt(n-1) \; UI \; rms$$

Therefore in a DS-3 network simultaneously synchronizing 19 nodes to a 20th reference node, the residual timing error is, with 95.5% (2σ) probability less than $2 \times 4.18 \times sqrt(19) \times 22.3 = 813$ nsec.
This implies that field precision of under a microsecond should be feasible. Note that the desired timing uncertainty and the number of nodes to be simultaneously synchronized can be traded-off.

4.2 Closed Time-Transfer Paths

An interesting extension is to arrange the time-transfer path in a loop by merging the *loop* and *trigger* sites. For example, in Fig.1 the path could be extended from node 1 to node 1 with node 3 performing both *trigger* and *loop* code functions. A closed path can integrate the special *trigger* and *loop* functions at one site which may house the NOC and/or a high accuracy time-of-day reference. All other DCS nodes can then use simplified hardware to support only the *intermediate* node role. This also permits the time-transfer process to be performed in both senses of travel of the m-code. It can be shown that this reduces the maximum residual timing error in a chain of nodes by a further sqrt(2), with respect to the one-pass method. If intermediate nodes select the results of the two transfer operations which involved the smallest local $C_i(t)$.

V. SUMMARY

We have described a technique for high precision distribution of time information in networks of DCS machines. The method was specifically devised to exploit the inherently fine time resolution of the wideband carrier signals to which a DCS has direct access. The resulting time-transfer scheme is a *master-slave*, *multi-site*, implicitly delay compensated, non-hierarchical method with a theoretical precision as high as one bit time at the carrier rate. The practical precision of the method is limited by the delay asymmetries between opposite directions of the selected transmission paths. However, a worst case estimate of delay asymmetry (including jitter effects) suggests that time-transfer errors of under one microsecond should be obtainable. The method is most easily supported by an n/n space-switching DCS which has equal delay on all paths through its core and has a bridging connection feature although other DCS properties can be accommodated through variations on the basic method. A circuit module for DCS hardware support of the time-transfer function was outlined. Measurement or improved estimates of characteristic span delay asymmetries is recommended to refine the preliminary performance estimate.

References

[1]   J. Biran, "Synchronous transmission", contribution to ECSA T1 standards project, T1X1A/86-179, Nov. 1986.

[2]   American National Standard for Telecommunications, "Digital hierarchy optical interface rates and formats specification", ANSI Doc. T1.10-1900, Draft, Mar. 1988.

[3]   J. Biran, "Procedure for time synchronization in Sonet", contribution to ECSA T1 standards project, T1X1A/86-480, Nov. 1986.

[4]   W.C. Lindsey, et al. "Network synchronization", Proc. IEEE, vol. 73, no. 10, pp. 1445-1457, Oct. 1971.

[5]   C.B. D'Espen and R.J. Kalpinski "Dissemination of system time", IEEE Trans. Commun., vol. COM-21, no. 5, pp. 805-824, 1972.

[6]   W.C. Lindsey and A.V. Kantak, "Network synchronization by means of returnable timing system", IEEE Trans. Commun., vol. COM-16, no. 6, pp. 892-896, 1978.

[7]   J. Yamato, M. Ono, and S. Ueno, "Synchronization of a PCM integrated telephone network", IEEE Trans. Commun. Technol., vol. COM-16, no. 1, pp. 1-11, 1968.

[8]   W.C. Lindsey and A.V. Kantak, "Mathematical models for time transfer networks", Proc. Int. Conf. Commun., Toronto, Canada, Jun. 1978.

[9]   W.C. Lindsey and H.J. Chu, "Mutual synchronization of two oscillators", NTC '82, Galveston, TX, Nov. 1981.

[10] P. Okonkwo and P. Dworiak, "Time synchronization and ranging system", ICC '82, Philadelphia, PA, Jun. 1982.

[11] C.C. Corrah, "Measurements via precautionary conditions", Proc. IEEE, vol. 74, no.1, pp.161-165 1986.

[12] B. Devoe and S. Lechheim, "The SIRIO-1 timing experiment", Alta Freq. (Italy), vol.51, no.5, pp.313-314, 1982.

[13] R.P. Vespers, "Time synchronization master station and remote station system", Canadian patent no. 7,116,164, Dec. 1982.

[14] Manitoba Telephone System, "Distributed network synchronization system", Canadian patent no. 1,157,119, Dec. 1983.

[15] H.A. Stover, "Improved time distribution for synchronous digital communications network", Proc. 8th Annual Precise Time and Time Interval (PTTI) Application and Planning Meeting, U.S. Naval Res. Ctr., Nov. 1976, pp. 161-184.

[16] D.O. Down, "Restoration of broadband fibre facilities (Bell Canada)", Fiber-Sat '78 Conference Record, pp.276-307, Vancouver B.C., Sept. 1986.

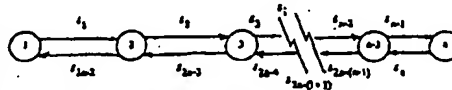## Appendix A : Analysis of Time-Transfer Method

Figure A.1   Nodes and Spans in Time-Transfer Path

n = total number of DCS nodes in time-transfer operation
$R_i(t)$ = numerical contents of real-time clock at site i at time t
$C_i(t)$ = numerical contents of a compensating counter at site i at time t ; $i = 1 ... n$
δ_i = propagation delay of ith unidirectional transmission span ; $i = 1 ... 2n-2$
ϵ_j = $\delta_{2n-(j+1)} - \delta_j$ = delay asymmetry ; $j = 1 ... n-1$ of jth bidirectional span
t = clock rate (Hz) of the carrier signal (i.e. DS-3 or STS-1) used for the time-transfer path

In Fig. A.1, node 1 is the *trigger* node, node n is the *loop* node and nodes 2...n-1 are intermediate nodes. The path from node 1 to node n is the t1 path and the reverse direction is the t2 path. At time $t=0$, node 1 transmits the unique m-code sequence which will be detected at one specific bit time at each decoder.

i.e.)   site 1 transmits

site 2 detects
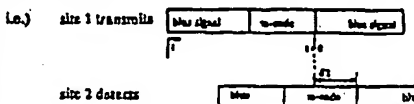
'te i will subsequently detect, on the t1 path, the m-code sent by node a at time t_{1j}:

$$t_{1j} = \sum_{j=1}^{i-1} \delta_j \qquad \text{A.1}$$

As each site detects the m-code on the t1 path, it starts its counter;

$$C_i(t) = \begin{cases} 0 & ; t < t_{1i} \\ \text{Int}[(t \cdot t_{1i})^* f] & ; t \geq t_{1i} \end{cases} \qquad \text{A.2}$$

Site n will detect the m-code at time $\quad t_{1n} = \sum_{j=1}^{n-1} \delta_j$

Immediately upon recognition of the m-code on the t1 path, node n appends the sampled value of its real-time clock to the bit stream. The m-code, followed by the data $R_n(t_{1n})$, now propagates via the loopback at node n into the return or t2 path.

io)   site n: [bias | m-code | δ bits]
                                           t_{1n} data for $R_n(t_{1n})$
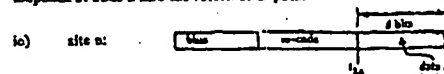
The appended data, $R_n(t_{1n})$, is the contents of the real-time clock register at site n at the instant t_{1n}. The R_n data requires a constant β clock cycles to transmit. (In general β may include the additional processing and transmission time required for error protection or other coding of the time data.)

As the m-code plus clock data propagates along the t2 path, it again encounters each DCS node. The time at which node i detects the m-code on the t2 path is designated t_{2i}

for site k $\qquad t_{2i} = t_{1n} + \sum_{j=n}^{2n-(i+1)} \delta_j \qquad \text{A.3}$

At t_{2i}, DCS node i stops its counter, $C_i(t)$, giving the complete function describing $C_i(t)$ as:

$$C_i(t) = \begin{cases} 0 & ; t < t_{1i} \\ \text{Int}[(t \cdot t_{1i})^* f] & ; t_{1i} \leq t < t_{2i} \\ \text{Int}[(t_{2i} \cdot t_{1i})^* f] & ; t \geq t_{2i} \end{cases} \qquad \text{A.4}$$

At site i at time $t_{2i} + β$, all of the clock data from site n has arrived and node i then latches the data field following the returning m-code. Node i then computes the sum $R_n(t_{1n}) + C_i(t_{2i})/2 + β$, and the result is loaded into the local clock register which is fed with the local DCS frequency source. If these sources are not synchronous, local clock registers will drift apart at a rate depending on the local clock stability and this will determine when another time-transfer operation is required to re-align all time-of-day clocks at one instant in time. To demonstrate that after time $t_{2i} + β$, node i has nominally the same clock register contents as node n, consider that at the instant $t_{2i} + β$:

$$R_i(t_{2i} + β) = R_n(t_{1n}) + β + \frac{1}{2} C_i(t_{2i} + β) \qquad \text{A.5}$$

and because the C_i counter was stopped at t_{2i}

$$C_i(t_{2i} + β) = C_i(t_{2i}) = \text{Int}[(t_{2i} - t_{1i})^* f] \qquad \text{A.6}$$

Therefore, substituting A.3 and A.1 for t_{2i} and t_{1i} respectively;

$$R_i(t_{2i} + β) = R_n(t_{1n}) + β + \frac{1}{2} \text{Int}[(\sum_{j=1}^{n-1} \delta_j + \sum_{j=n}^{2n-(i+1)} \delta_j - \sum_{j=1}^{i-1} \delta_j)^* f] \qquad \text{A.7}$$

At the same instant in absolute time at site n:

$$R_n(t_{2i} + β) = R_n(t_{1n}) + β + \text{Int}[(\sum_{j=n}^{2n-(i+1)} \delta_j)^* f] \qquad \text{A.8}$$

Subtracting A.8 from A.7, allows comparison of clock register contents at node i and node n,

$$R_i(t_{2i} + β) - R_n(t_{2i} + β) = \frac{1}{2} \text{Int}[(\sum_{j=1}^{n-1} \delta_j - \sum_{j=n}^{2n-(i+1)} \delta_j - \sum_{j=1}^{i-1} \delta_j)^* f]$$

$$= \frac{1}{2} \text{Int}[(\sum_{j=1}^{n-1} \delta_j - \sum_{j=n}^{2n-(i+1)} \delta_j)^* f] \qquad \text{A.9}$$

If we now define n-1 span delay asymmetries, ϵ_j, from the pair-wise association of the 2n-2 unidirectional spans shown in Figure A.1 as follows

$$\delta_j = \delta_{2n-(j+1)} + \epsilon_j \qquad \text{A.10}$$

Substituting A.10 into the first summation term in A.9 gives

$$\sum_{j=1}^{n-1} \delta_j = \sum_{j=1}^{n-1} \delta_{2n-(j+1)} + \sum_{j=1}^{n-1} \epsilon_j \qquad \text{A.11}$$

It can be shown that the series $\delta_{2n-(j+1)}$ enumerated with j = 1..n-1 produces all elements of the series δ_j enumerated over j=n ..2n-(j+1) and therefore equation A.11 can be rewritten as

$$\sum_{j=1}^{n-1} \delta_j = \sum_{j=n}^{2n-(i+1)} \delta_j + \sum_{j=1}^{n-1} \epsilon_j \qquad \text{A.12}$$

Now substituting equation A.12 into A.9 gives the difference between clock register contents at node i and node n as;

$$R_i(t_{2i} + β) - R_n(t_{2i} + β) = \epsilon_i(t_{2i} + β) = \frac{1}{2} \text{Int}[(\sum_{j=1}^{n-1} \epsilon_j)^* f] \qquad \text{A.13}$$

where $\epsilon_i(t)$ denotes the error between the node i clock register and the node n clock register at time t. We call $\epsilon_i(t_{2i} + β)$ the residual time error of the time-transfer process.

A.13 shows that for the ideal case of perfectly delay-balanced propagation paths ($\epsilon_j = 0$; 1 ≤ j ≤ n), the contents of the real-time clock register at node i are identical to the clock contents at node n, after the time-transfer operation. In all other cases, the residual time error is one half of the net delay asymmetry between a given node i and the loop node n. If ϵ_j is a zero-mean gaussian random variable with standard deviation $σ_e$, in unit intervals (UI), and delay asymmetries are uncorrelated span to span, then for the ith node in a time transfer path of n nodes;

$$E[\epsilon_i] = 0 \qquad \text{A.14}$$

$$σ_{\epsilon_i} = \sqrt{(n-1)}/2 * σ_e \qquad \text{A.15}$$

47.2.5.

The Scalable Coherent Interface Project (SuperBus)

David B. Gustavson, Stanford Linear Accelerator Center
David V. James, Hewlett Packard
John Moussouris, MIPS
Paul Sweazey, National Semiconductor

Abstract

Our goal is to develop an interface standard for very high performance multiprocessors, supporting a coherent shared-memory model scalable to systems with up to 64k nodes. This Scalable Coherent Interface (SCI) will supply a peak bandwidth per node of at least 1 GigaByte/second, with a total flux of N GigaBytes/ second in a system with $N \le 64K$ nodes. The standard will facilitate assembly of processor, memory, I/O, and bus adapter cards from multiple vendors into massively parallel systems with throughput ranging beyond 10^{11} operations per second.

The SCI standard will encompass two levels of interface. The physical level will specify electrical, mechanical, and thermal characteristics of connectors and cards that meet the standard. The logical level will describe the address space, data transfer protocols, cache coherency mechanisms, synchronization primitives, and the control and status registers used for initialization, exception handling, and error recovery.

Introduction

After working on some of the fastest state-of-the-art computer buses, we have concluded that a radical new approach will be required to achieve the kind of performance we dream of for the next generation of computing machinery.

Distance and propagation delays impose fundamental limits on the time required to transfer data on present buses. In asynchronous buses, the limit is the time needed for a handshake signal to propagate from sender to receiver and for a response to return to the sender. In synchronous buses, it is the time difference between clock and data signals which originate in different places.

Signal distortion and noise created by practical compromises in real systems are often as significant as these fundamental space-time limits. The ideal transmission lines we imagine on our

Scalable Coherent Interface   *August 22, 1988*   *Rev. 14,   Page 1*

backplanes are always disturbed by T's and stubs where connectors attach (see Figure 1 and Figure 2), and the bus transceiver circuits on the plug-in modules are also less than ideal. The variations in loading as modules of different kinds are inserted in various numbers make it impossible to terminate bus transmission lines correctly for all conditions. The number of modules permitted has to be traded off against tolerance to signal reflection effects, connector spacing, etc. Other sources of noise include crosstalk, power distribution IR and LdI/dt, and non-ideal ground planes. Though BTL (Futurebus) and ECL (Fastbus) signalling work much better than the more common technologies, they still have practical limitations.
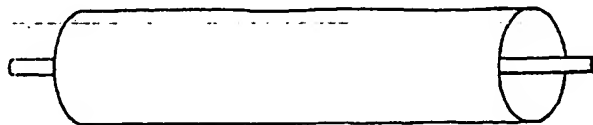
Figure 1: Ideal coaxial transmission line.
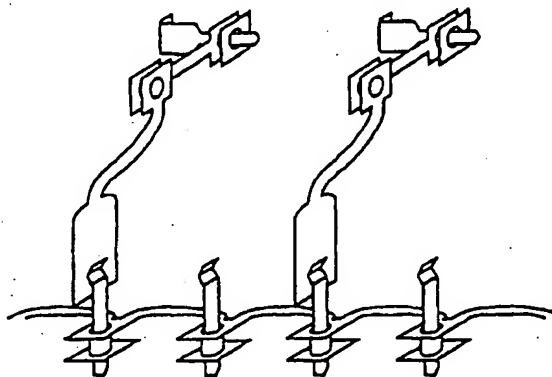
Figure 2: Actual bus transmission line.

Even if the bus were ideal, it can still be used by only one sender at a time. Hence a bus becomes a bottleneck in multiprocessor systems, when multiple processors need more cumulative bandwidth (flux) than the bus can provide.

The best modern buses have pushed hard against these limitations. For example, Futurebus adopted a new, improved transceiver technology and devised distributed cache protocols which can greatly reduce multiprocessor bus traffic. Fastbus adopted a multiple-bus-segment scheme with

dynamic interconnections for increased parallelism, an efficient block transfer protocol, and a pipe-lined block transfer mode which eliminates handshake delays. Its speed is ultimately limited only by signalling bandwidth, but in practice skew (differences in effective propagation velocity among the various parallel signals) sets the limit. And not to be outdone, both Futurebus and Fastbus are busily incorporating each other's best features... Yet they dare not change too radically, because they have to maintain compatibility with existing devices. But without a radical change, we can only edge closer to the space-time, noise, and flux limits delineated above. To get beyond these limits, we need to change the fundamental paradigm.

Looking for new approaches to solve these problems, one of us (Sweazey) proposed an IEEE Computer Society (Microprocessor Standards Committee) "SuperBus" Study Group, to determine whether it might be practical to create a new standard which would benefit from the experience gained with the existing buses and make a major improvement in performance. The goal was to achieve Gigabyte/second bandwidth while supporting many processors.

The SuperBus Study Group worked actively (typically two or three meetings a month) for less than a year before concluding that these goals are feasible. In fact, encouraged by our early suc-cesses, we were inspired by a suggestion of Paul Borrill to escalate the goal to a peak bandwidth of N GBytes/second in a system containing N nodes, where N can range as high as 64K. The Study Group is now applying for IEEE Project status, with the intention of generating a new standard.

Note, however, that even if this ambitious goal is met in record short time, current designers ought in nearly every case to use the existing standards, because the essential VLSI support chips may not be available until a considerable time after the new standard is finished and stable!

What are the changes in the physical and logical paradigms which make the SuperBus Study Group so optimistic? The space-time limitations of traditional buses can be overcome by abandon-ing bus structures in favor of point-to-point interconnects. Each SCI node is connected to the rest of the system through a single pair of unidirectional links as shown in Figure 3. Skew can then be minimized by source synchronous clocking: i.e., a strobe generated at each source accompanies all data bits through matched drivers, traces and receivers.

Susceptibility to signal distortion and noise can be reduced by differential current-steered sig-nalling with controlled edge rates. The signals are unidirectional and uninterrupted, so that the net

Figure 3: Nodes are connected by two unidirectional links.
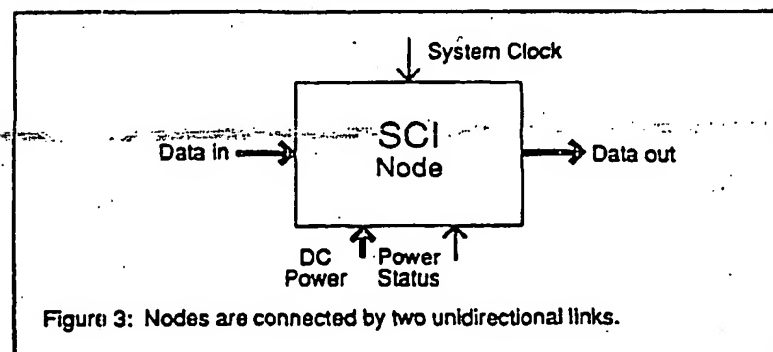
current flow through each link is constant. Each differential pair travels from a single driver to a single receiver through pins and traces that are physically adjacent.

Even using currently available ECL circuitry with this approach we think we can achieve 500 MHz signalling rates, allowing 1 Gigabyte/second bandwidth on a 16-bit-wide link. Such a narrow high-speed link facilitates VLSI implementations and the wiring of switching networks.

Support of this physical paradigm requires radical changes in the logical paradigm as well. Since each link is unidirectional, handshakes must move up to a higher level of protocol, much as computer networks control the flow of data packets by returning response packets instead of by handshaking individual bits. For example, a processor-to-memory read operation is split into two separate packets: a request packet provides the address, and sometime later a distinct response packet supplies the data. Figure 4 shows the overall structure of a typical SCI packet.

| Target | Source | Control | Address | Data... |

Figure 4: A typical SCI packet.
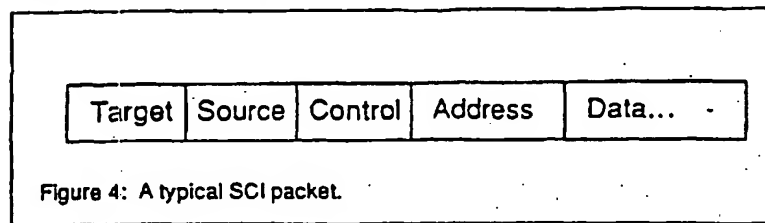
To overcome the flux limitations of conventional buses, SCI must support thousands of nodes communicating through a switch network in parallel. To reduce contention through the switch, cache memories must stage local data at the nodes, while the system maintains a coherent image of shared memory. Traditional cache coherency mechanisms rely on broadcasting and eavesdropping, which cannot be used in our highly parallel environment. Directory based methods can be used instead so that coherency traffic only involves those nodes sharing a given data item.

In general, our goal of ultimately supporting thousands of processors contributes the primary constraint on our designs; in practice the most powerful constraint we face as we consider alternate architectures: we refuse to consider any mechanism which scales badly as the system grows larger.

Hence our new name: Scalable Coherent Interface. We don't think of it as a bus anymore (though it will certainly be Super). We give primary importance to scalability. We want coherence for efficiency reasons. We want to specify an interface to standard modules, to provide economies of scale for module production and an economical upgrade path for the user. The interface specification must include protocols, signals, connectors, geometry, power, and cooling.

Configurations

Figure 5 shows the essential organization of SCI. The details of the interconnection are concealed from the nodes. Many different structures could be used inside the SCI 'blob': full crossbar switches, optimized N-way switches, rings, buses, and arbitrary combinations of these connected together.

Figure 5: The organization of an SCI system.
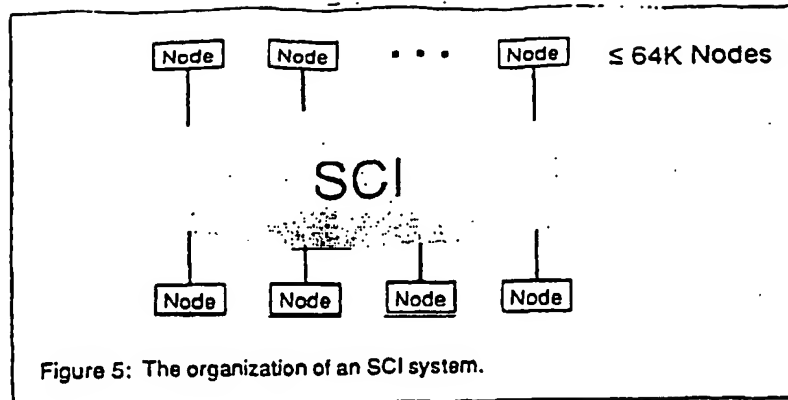
The real world is more complex, including pre-SCI systems built from various bus standards.
An important goal of SCI is to make it possible to interface these systems to SCI and thus (indi-
rectly) to each other. There will be some limitations, of course, because the older systems will
probably lack some desirable features which are not easily simulated by an interface. Nevertheless,
some proposed SCI features are harder to interface to than others, and we weigh these considerations
in our architectural decisions. Another practical consideration is the need to configure clusters of
SCI systems. Figure 6 shows a typical case, where two SCI systems which were built independently
are connected to each other and to several independent subsystems built out of various standard
buses.

Figure 6: SCI is a generalized interconnection which can combine other
bus systems and SCI subsystems.
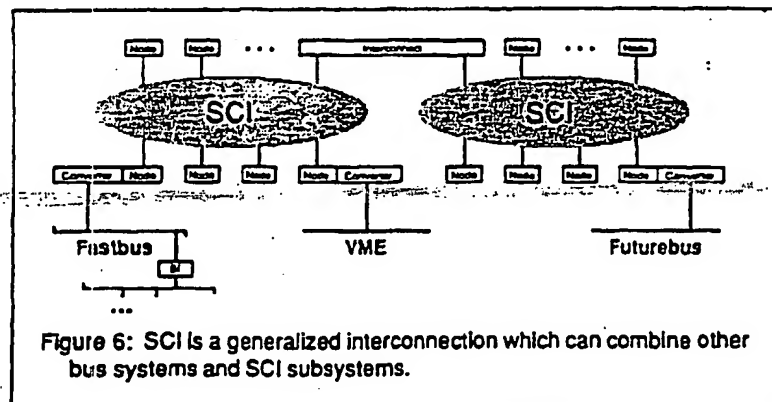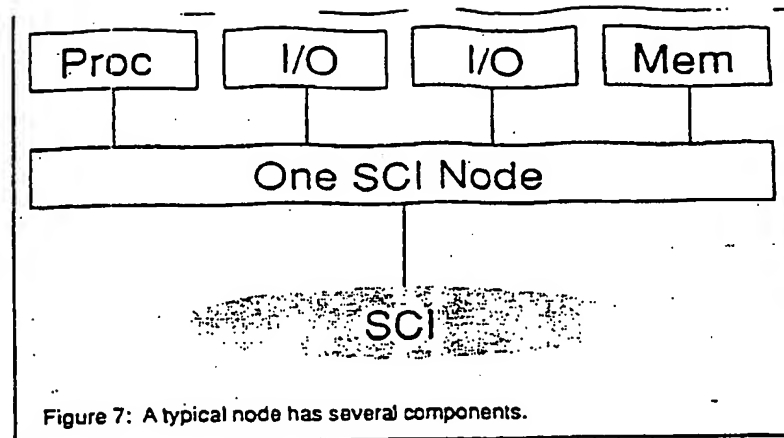
Figure 7: A typical node has several components.

Figure 7 shows a typical node. The node may contain a variety of processors, I/O device adapters, and memory. The control registers for each part are accessed via standardized addresses.

Architectural Approach

The primary elements of the SCI architecture are the address space, data transfer protocols, synchronization facilities, cache coherence mechanisms, exception handling, and initialization procedures.

Poor addressing mechanisms have crippled more computer families than any other single flaw. Addressing is always a compromise between elegance and simplicity on the one hand and speed and cost on the other. Our preference now is for a flat 64-bit address space, with a 16-bit node ID and a 48-bit offset, interpreted as a byte address, for use within each node (Figure 8). The 16-bit node ID limits our systems to 64K nodes, which seems a little risky until one realizes that each node can be a multiprocessor itself. The node ID has to be decoded at very high speed, which argues for keeping it as short as possible.
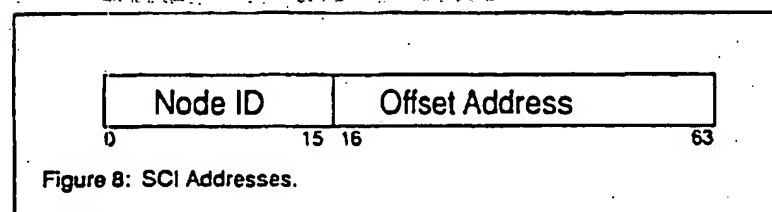
Figure 8: SCI Addresses.

The 48-bit offset provides for 256 Terabytes of physically addressed memory and registers in each node. The virtual address space of an SCI system is not tied to this physical limit, however. Our goal is to make SCI protocols powerful enough to allow a variety of virtual memory management schemes to be implemented efficiently under software control.

In keeping with the unidirectional packetized nature of SCI each data transfer has a unique source node and a unique target node. There are no atomic transfers that require information flow in

Request	Target (Mem_ID)	Source (Proc_ID)	Command (READ_Req)	Address (Memory_Offset)	
Response	Target (Proc_ID)	Source (Mem_ID)	Command (READ_Resp)	Status (OK)	Data (Data)

Figure 9: Packets for a typical SCI read operation.
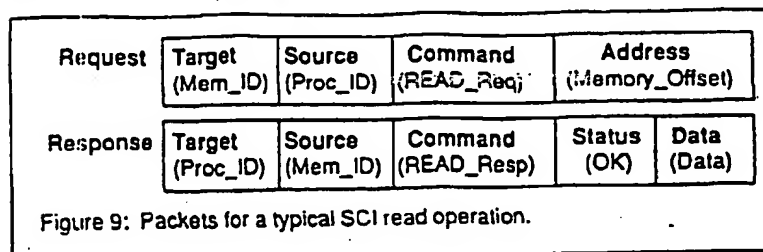
both directions. For example, to read memory (see Figure 9), send a packet to the memory requesting data from a particular address. While the memory is looking for the data, SCI is freed for other operations. When the data is found, the memory sends it back in another packet. This mechanism is used for inter-processor communication too; in fact, we expect most memory to be intimately associated with some processor.

These unidirectional transfer protocols entail special problems for 'read-modify-write' operations that are traditionally used to synchronize processes and processors (to implement the equivalents of semaphores or rendezvous, for process scheduling and resource management). Our solution is to implement synchronization operations atomically inside of a single node which currently contains the synchronization variable. The atomic operation typically involves locking the data so no one else can change it, saving its current value, changing the data in some way, unlocking the data again, then returning the saved data value to the requestor. Performing this complex sequence inside a single node is much more efficient than using multiple operations that lock up the SCI switching circuitry.

Cache coherence protocols are used to achieve the performance advantages of fast local cache memories, while maintaining a flat shared-memory model. The trick is to make sure that the system never allows multiple copies of the same data to become different (inconsistent or incoherent) as a result of one processor's changing it without the others' knowledge. For example, on Futurebus every cache monitors the bus traffic, looking for operations which might affect the validity of its own data. In addition, each cache has to report on the bus any change to its data, unless it knows no other cache contains that data.

Unfortunately, this eavesdropping mechanism requires every cache to see every data transfer on the bus, and that is incompatible with our goal of many simultaneous independent communications. Though it is possible to extend this scheme somewhat by using clever interfaces between multiple buses, it does not scale well for really large systems.

Scalable Coherent Interface   *August 22, 1988*   *Rev. 14,   Page 7*

E.D.VA. CA 00-524
1 207272

863 FH PG 0788

DRAFT

SCI uses a 'directory-based' coherence mechanism instead. The cache controllers and the memory cooperate in order to keep track of who has permission to write a particular piece of data (one at a time, please!) and maintain a list of everyone who has gotten a copy of that data, so they can be notified if it changes. This sounds complicated at first, but it seems to be manageable. It sounds inefficient, too, with all the notifications being sent out, but that turns out not to be too bad either—the number of notifications is less than or equal to the number of data accesses. The inefficiency is at most a factor of two, which is insignificant compared to the cost of broadcast mechanisms.

We are still working on error handling and initialization. There are many ways to handle these which seem to work without serious scaling problems, but it will take some time to decide on the best strategies. The emerging P1394 SerialBus has a high-level architecture very similar to SCI. It will be incorporated into the SCI standard to provide a redundant low-speed initialization and diagnostic path.

Data Transfer Protocols

What transactions should SCI support? Let us consider a variety of typical situations to see what might be useful. To access control registers in SCI or on foreign buses, an assortment of small read, write, and lock operations is required. To support cache-line oriented transactions, larger block operations are needed.

We limit the maximum block transfer length to 256 bytes. Longer blocks would only increase the efficiency a small amount, because the packet overhead is only a few percent of 256 bytes. Furthermore, longer blocks tie up switch resources, block other traffic, and increase the average latency.

We support only a fixed set of block lengths (powers of two), and require that blocks be aligned on a corresponding power of two memory address. This simplifies cache logic, making it easy to determine which cache entries will be affected by the transfer. When necessary, odd length or misaligned transfers are broken up into short pieces at the beginning and the end with aligned blocks in the middle.

There is little efficiency to be gained by defining explicit short transfers, because we always have the packet overhead anyway. We make short transfers special cases of the 16-byte block transfer, as shown in Figure 10.

	selected bytes	

A16   A16+16
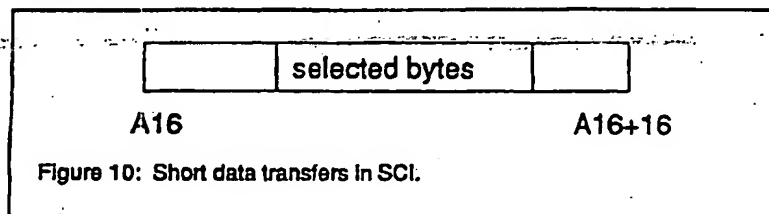
Figure 10: Short data transfers in SCI.

Thus we support operations on 32, 64, 128, and 256 byte data blocks, and 1–16 byte subsets of the 16-byte block, starting and ending at arbitrary points in the aligned 16-byte field.

SCI provides synchronization mechanisms needed for implementing semaphores and allocating resources in a multiprocessor system. The fundamental lock primitives are all of the form of an

atomic read-modify-write. Each primitive sends a request packet containing new data to a given memory cell. The operation performed is either to add, store, or compare and conditionally store the new data in the memory cell. In order for the requester to determine the status of the lock action, the old data is always returned in the response packet. Hence the lock operations are named Fetch_Add, Swap, and Compare & Swap. We also propose a primitive List_Insert which is a variant of Compare & Swap useful for atomically inserting a new entry at the head of a linked list.

Inside a multiprocessor system, memory locks will generally be performed on data which is temporarily locked in a data cache. However, explicit lock operations are required if it is necessary to perform these operations on data not in the processor's cache. For example, when combining networks are used to reduce synchronization bottlenecks, locks should not be cached.

All of these bus operations are summarized in Figure 11.

Function	Sizes	Description
Bread	32,64,128,256	Read Block
Bwrite	32,64,128,256	Write Block
Sread	Selected-16	Read contiguous subset of Block-16
Swrite	Selected-16	Write contiguous subset of Block-16
Fetch_Add	Selected-16	Lock Primitive
Swap	Selected-16	Lock Primitive
Compare & Swap	Selected-16	Lock Primitive
List_Insert	Selected-16	Lock Primitive

Figure.11: Summary of SCI Operations

Coherence Protocols

The fundamental problem of cache coherence occurs when a processor attempts to store data into a memory cell that is already cached by one or more other processors. All the cached copies must be invalidated before any other node is allowed to write the item, or updated when the item is written. Because we cannot implement broadcast or eavesdropping mechanisms in a scalable architecture, we must use a directory scheme to keep track of all readers of a given item.

Our present proposal is to maintain a linked list of the nodes currently sharing a given data item by means of pointers stored in the node caches themselves. Each coherency block in a cache has a forward pointer and a backward pointer to neighboring nodes in the list sharing that item (see Figure 12). The memory card associated with that item is ultimately responsible for initializing the list and directing requesters to the node whose cache contains the head of the list. The virtue of putting the pointers in cache is scalability: we automatically have more room for directory information as we add more nodes or larger caches. The disadvantages are increased cache tag size and complexity. These disadvantages have motivated the working group to continue its study of alternative proposals.

Scalable Coherent Interface　　　*August 22, 1988*　　　*Rev. 14,　Page 9*

The operation of the underlying protocols can be illustrated as follows. Suppose a large number of caches are sharing a given data item as shown in Figure 12. A new node W wishes to modify a data item. First it issues a request to read memory and acquire private ownership. The memory read detects the existence of other readers. In addition to returning the data, the memory returns the pointer R[0] to the head of the list of others sharing the data. The memory also updates its own pointer to make W the new head of the list. The list of sharing processors is followed, invalidating the copies and removing each entry from the list. When that process is complete, the current requester has exclusive control over the value of the data, which can be modified at any time.

· Figure 12: Sharing-list pointers for coherence with many readers.
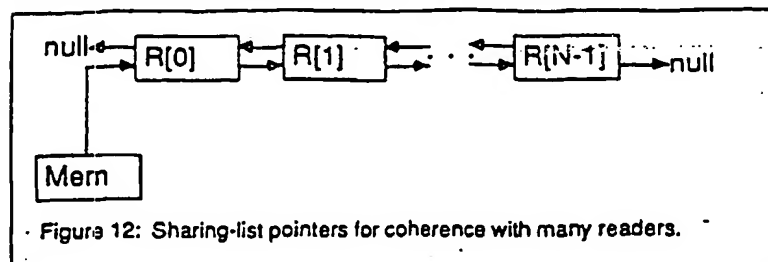
Certain special cases can be optimized, such as the producer-consumer relationship between a single writer and reader shown in Figure 13, where the rights to the data must rapidly bounce back and forth between the two. The arrows in the figures show the pointer structure which is maintained.

Figure 13: Sharing-list pointers for coherence with one writer, one reader.
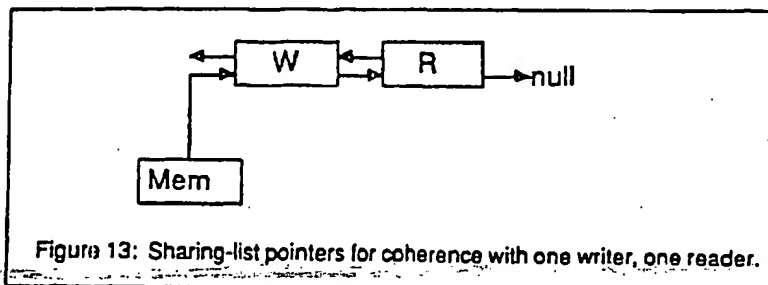
DMA and Message Passing

To maintain the high computational throughput of a 64K-node SCI system, a DMA architecture is needed that can efficiently support a corresponding large number of I/O devices. We are proposing a DMA architecture which specifies control registers and standard DMA commands. The basic scheme is to load a DMA command program into memory, and to write a pointer to this program into the DMA device control registers. Included in each sequence is a pointer to a memory area which is to receive the status upon completion.

When a command sequence is completed, a new status block is updated and inserted into a processor interrupt service list. Writing to the processor's interrupt control register generates an interrupt which initiates processing of this service list. By reading the service list, the processor can determine what action is required without polling each of the active DMA devices. Thus this architecture gains efficiency both by reducing the number of interrupts generated and by speeding the processing of each one.

Message passing is another important multiprocessor feature supported by SCI. The SCI DMA status-lists and interrupts provide an efficient way to transfer message packets and notify the recipient. Though SCI gives high priority to supporting a shared memory model, some systems may not decide to share all memory. The Source ID present in every packet provides the information needed to implement selective access permissions. I.e., a node may decide whether or not to allow memory access based on the address requested and the ID of the originator. This can be used to provide the safety generally associated with message-passing operation. The tradeoff between visibility and protection or concealment will be a system implementor's choice.

SCI also supports 'tag' bits for both address and data fields in a packet (one tag bit per 16 data bits). These tag bits can be used for a variety of purposes. One potential use is for implementing 'futures', where one task may try to read data which has not yet been written by another. When the tags label the data as invalid, the reader is suspended until the data and tags are written. Tags provide the synchronization between tasks on a data-item by data-item basis, as opposed to the more common block-by-block basis using one semaphore per block. Tags have also been used to identify data types in special-purpose processors (e.g., for Lisp).

Physical Level

Our goal is 1 GigaByte per second for each of $N \leq 64K$ processors. Providing an independent high-performance data link for each communication is not a trivial problem. Of course, we expect to see a variety of implementations which make compromises to bring the costs down by sacrificing performance (typically by reducing the number of simultaneous independent paths), but we do not want to build this sort of compromise into the SCI definition.

Data path width is very costly in switching networks. The building blocks used to implement switches are VLSI integrated circuits with lots of pins: a chip which can connect four ports to four other ports in any permutation (4-by-4) has eight ports, which requires 256 I/O pins for a 32-bit SCI implementation. Furthermore, a complete switch requires four times as many 2-by-2 chips as it would 4-by-4 chips, with twice as many total pins. Given the limitations of practical packaging technology, we must use narrow signal paths at the highest possible speed to achieve our performance goals.

At these speeds, one has to be very careful with signalling technology. Every signal exists in a transmission-line environment, and reflects off every discontinuity in its path. Connector pins become complicated discontinuities with large inductances and capacitances to adjacent pins. We have to account not only for the signal current but also for its return path ("ground") as it completes its circuit. A 'ground' pin going through a connector can easily be part of a resonant circuit at these high frequencies—the good grounds we can achieve (with care) at low frequencies become faded dreams.

At first these practical problems seemed insurmountable, but now we think there is a workable solution. The key was realizing that we don't need to use bus technology: we want point-to-point

DRAFT

links. It gets even easier if we use separate links for in and out directions. For point-to-point unidirectional links, we can use differential signalling, where each signal uses two wires which are always in complementary logic states. The receiving circuit considers only the sign of the difference of the signal voltage between the two wires, ignoring signal magnitude. This method is relatively immune to noise, which generally has the same effect on both wires (and thus can not change the sign of their voltage difference) if the wires are kept close together.

Complementary signal outputs are a standard feature of Emitter-Coupled Logic (ECL), the high-speed technology long used by the fastest computers and available from many vendors in modern designs which offer very high performance. Using readily-available components, we believe we can achieve transmission rates of 500 Mega-transfers per second, so with a 16-bit-wide data path we can reach our 1 GByte/second goal.

We could do even better than standard ECL if we used complementary current-driving outputs (ECL does this internally, but converts to voltage drive near the output pins). Current drive just steers a constant current to one pin or the other of the complementary pair, maintaining constant current flow to the integrated circuits, or through any connector carrying such signals. Note that we assume unidirectional links—if we use a link bidirectionally, we have to turn off the drivers at one end before turning on the drivers at the other end, which requires cooperation between the ends (an arbitration mechanism) and introduces sudden changes in the net current flow, creating noise in the system.

We insist in fact that all signals in any link travel in the same direction. For example, we do not allow a receiver to send any reverse signal to tell the sender to slow down or to stop because the receiver cannot keep up. In systems which are large compared to the distance signals travel during a quarter of a clock period (about 150 millimeters—i.e. in any real system), the time delay between reverse signals and the corresponding forward signal depends so much on the particular connection path that the meaning of such signals becomes hard to interpret. Since such mechanisms do not scale well, we forbid them.

In addition to sending signals, a timing marker lets the receiver know when to capture the data. Typically this is a strobe signal whose edge transitions occur at a specified time with respect to the transitions from one data word to the next. With complementary signalling, both strobe transitions are equally usable, so the strobe frequency is the same as the data frequency.

In any real system, there will be small differences from one signal path to another within the link. These differences are called skew, and if the skew gets to be a significant fraction of the strobe period, it becomes impossible to determine when to store the data to ensure that all bits are really part of the same word. To achieve 2 nanosecond data rates, skew must be kept well below 1 nanosecond. Our method for achieving this is to make strobing source-synchronous at the individual link level. That is, a local strobe generated at the source end of each link will accompany data through matched drivers, traces, and receivers to the destination. We expect to provide a standard clock at one place in the system, which will be used to keep all data links operating at the same frequency, but the phase of this clock with respect to the data strobes will vary from place to place.

Skew is often the most significant limiting factor in the speed of parallel transmissions. Reducing it beyond a certain point becomes expensive and impractical. Where necessary, skew can be eliminated by including a strobe signal in each data signal line, ensuring that there are enough timing transitions for reliable data extraction. Manchester or group encoding are often used for this purpose. This sort of mechanism has long been used on each track of standard magnetic tapes, so the data from each track is first recovered independently and then combined with the data from the other

Scalable Coherent Interface   *August 22, 1988*   *Rev. 14,   Page 12*

tracks to complete a record. It appears possible to do this at SCI speeds, if necessary. These mechanisms work much better in a system where data is transmitted continuously in only one direction.

Figure 14 shows the signals seen by an SCI node. There are 16 bits of data, two bits of flag, and one bit each for parity and strobe, in each direction, giving a total count of 80 complementary signals. In addition, each node requires DC power, power status, and the system clock. The SCI modules can be initially reset to a known state by using the power status signals.
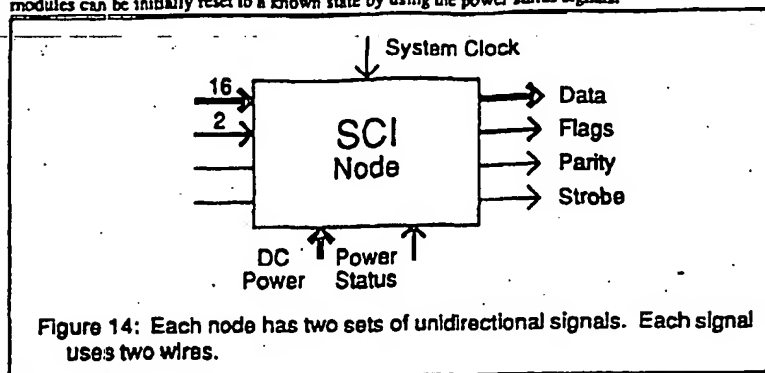
Figure 14: Each node has two sets of unidirectional signals. Each signal uses two wires.

Figure 15 shows the data format we have assumed for preliminary design studies. The extra bits marked with the question mark could perhaps be used for tags on address and data.

flag0 flag1	16 bits for data	Comment
0 0	«16:Don't care»	Idle cycle
0 1	«1:old»«15:Reserved for local routing»	Start
1 «?»	«16:Target Node ID» .	
1 «?»	«16:Source Node ID»	
1 «?»	«8:Transfer Code»«8:Request Sequence»	
1 «?»	«16:Address Offset Word 0»	
1 «1:bsy»	«16:Address Offset Word 1»	
1 «1:ack»	«16:Address Offset Word 2»	
1 «Longitudinal parity covers parity, flag 1, data but not flag 0»		?pHeader
1 «?»	«16:Data Word 0»	Optional
1 «?»	«16:Data Word 1»	
...		
1 «?»	«16:Data Word n»	
1 «Longitudinal parity covers parity, flag 1, data but not flag 0»		pData

Figure 15: Proposed request-packet data format for SCI links.

The strobe runs continuously, whether there is data to send or not. The flag bits show whether data is being transmitted and mark the start of the packet.

The variable-length data block follows the packet header, which contains the 48-bit address offset and the 16-bit Target Node ID which together form the SCI 64-bit address field. The first word, called Start, is used for local routing within SCI. The Target Node ID determines the route through the SCI system, based on information loaded into SCI nodes at initialization time. The Source Node ID provides the address needed for returning a response packet.

The Transfer Code specifies the operation to be performed. The Request Sequence number is used to differentiate each of the pending operations which have been generated by the same requester.

The «old» bit is used for marking and discarding undeliverable packets, or for garbage collection, in some implementations which might otherwise be vulnerable to endlessly circulating packets. The «ack» and «bsy» bits are used as part of the lowest-level flow control. If the receiver recognizes the packet but is temporarily out of buffer space, it sets «bsy» and returns the entire packet, for the sender to retransmit later. Alternatively, «bsy» could be signalled via a short packet rather than by returning the entire packet.

Block parity checking is currently assumed. One parity bit protects each word of data; one parity word also protects each block of words. Other forms of checking are also being considered, including the use of more sophisticated detection/correction schemes computed during packet formation; the timing (location) of «ack» and «bsy» status bits is also a subject of active discussions.

Lower-Cost Implementations

So far, we have considered SCI as a very general interconnection mechanism, presuming it to be a switched network of some sort in order to reach our goal of flux scaling linearly with the number of nodes. We hinted that some implementations might compromise this goal in favor of economy, using less than a full crossbar switch. In fact, we believe that some very low cost implementations are possible which still have interesting performance.

We think there is an attractive solution based on ideas presented to us by Manolis Katevenis. This is to use a parallel version of an insertion ring, where each node has a unidirectional link to its neighbor, with the last node connecting back to the first to form a closed loop. The bandwidth of the ring is shared by all the nodes, so one would not wish to put very many nodes on one ring. In the future, when node bandwidth requirements become comparable to our link bandwidth, one could make a system with many small rings interconnected by specialized repeater nodes. In fact, one could implement a variety of interesting switch networks in this way.

Though rings look quite different from generalized switch networks, it seems that the information needed for routing packets is compatible, so that the same modules could be used in either environment without change. If necessary, a connector pin could tell the module which kind of connection it has so it could modify its behavior slightly if that should prove desirable.

Although there is no obvious limit on the number of boards which could share one ring, the average bandwidth per node will decrease with the size of the ring. The ring latency will also grow in proportion to the number of nodes. Each node must have several stages of registers in order to deskew and reclock the data bits. Furthermore, internal FIFOs will add to the latency when conflicting traffic is encountered.

Scalable Coherent Interface   *August 22, 1988*   *Rev. 14.   Page 14*

As long as the idle-ring latency is comparable to .00 nanoseconds (typical large-memory access time) we think the ring will be acceptable in performance. When traffic is high, delay will be dominated by waits for access in any bus structure, at least as bad as the latency for the ring. Assuming 2 ns per clock, four register stages per module, and 8 modules, we get 64 ns ring latency, assuming an idle ring.

The ring does not need an arbitration mechanism, but does need an allocation mechanism. Our strategy is that begging forgiveness is more efficient than asking permission. Send a packet out, whenever the output link is available. Buffer any incoming data which arrives while our packet is being sent. Empty this buffer before sending another packet.

How Does the Ring Work? The model is shown in Figure 16:
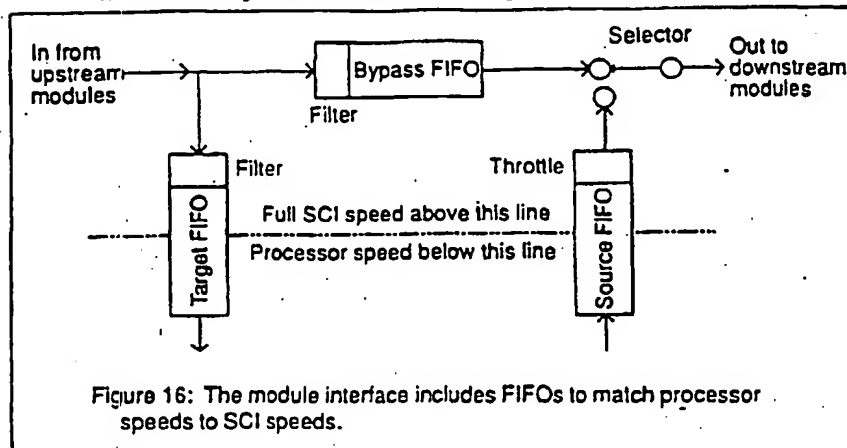
Figure 16: The module interface includes FIFOs to match processor speeds to SCI speeds.

The source FIFO should be big enough to hold our largest transmission packet. Because we may have to fill it slowly, it has to contain the entire packet before its (high-speed) transmission begins.

The target FIFO stores only packets addressed to this node. They enter at high speed while exiting at normal microprocessor speed, so there is the possibility this FIFO might overflow. When there is not enough room to hold a packet, «bsy» is returned to the sender, requesting retransmission.

The bypass FIFO accumulates incoming bits while this node is transmitting. The idea is that we never start transmitting until it is empty, so if the bypass is as big as our own maximum packet length it cannot ever overflow.

The key to successful and efficient operation is the throttle algorithm. In the simplest implementation, we never start a transmission from our source FIFO unless the bypass FIFO is empty (and thus has room for the amount of data we are about to transmit).

SCI-22Aug88-doc1-p16   DRAFT

When the sender receives its own successfully delivered packet, it accumulates it in the bypass FIFO, and then removes it. This scheme is simpler than a token ring, and is inherently fair. There is no need to worry about multiple tokens—several transmissions may begin at once without causing conflicts like they would on a bus structure. There is some obvious waste in this scheme, because the entire delivered packet continues around the ring back to the sender, using cycles which could have been used by others on that part of the ring. More efficient allocation algorithms, which allow these cycles to be used by others, are under active consideration.

What about the case where there is only one module who wants to transmit (multiple packets)? If the packet is shorter than the ring delay, cycles would be wasted if each packet is acknowledged before another packet is sent. We allow multiple packets to be sent as long as the bypass FIFO remains empty.

Conclusions

We have found an approach which seems promising. We avoid space-time problems by abandoning bus structures in favor of point-to-point links with source-synchronous clocking. We reduce signal noise and distortion problems by using differential unidirectional transmission. We avoid the flux shortage by an architectural approach which allows a very high degree of parallelism. Though there is still a great deal of work to do, we feel optimistic that this approach will bear fruit.

Current Status

The Study Group has been meeting monthly, with working task group meetings interspersed. We are now in the process of becoming an official IEEE standards working group. If you would like to participate, please contact David Gustavson, SLAC Bin 88, P.O. Box 4349, Stanford, CA 94309, USA, telephone (415) 926-2863.

We have a good first-pass draft Logical Protocols document, are now working on an I/O architecture document, and are in the early stages on the physical layer design.

SCI Meeting Schedule

Unless otherwise stated, meetings are all day, 9:30 AM-5 PM.

While this information is current as of August 22, 1988, it might be
prudent to confirm location and schedule before travelling...
call the host, or Dave Gustavson at 415-926-2863.

In odd-numbered months, our meeting will be on the Tuesday after the second Monday, i.e., the
day after the IEEE Computer Society's Microprocessor Standards Committee meeting. We will
generally use this formula for even-numbered months as well, but may make exceptions to take
advantage of relevant conferences etc.

August 26, 1988 9 AM I/O Registers Task Group
          (jointly with Futurebus P896.2 and Serial Bus P1394)
        1 PM SCI Physical Layer Task Group
          Connectors, signalling, packet format, etc.
        Hewlett Packard – 3U   Host: Hans Wiggers, 415-857-2433
        1501 Page Mill Road
        Palo Alto, California 94304

September 13   National Semiconductor Host: Paul Borrill, 408-721-7443
        Building 16
        2900 Semiconductor Drive
        Santa Clara, California
        (really on the south side of Kifer, a few hundred meters west of Lawrence—
        the main marketing building)

October 4   BUSCON/88-East
        9 AM–12 AM, Room 1A01
        Hotel Penta
        7th & 33d
        New York City
        Hotel: 212-736-5000

October 13   Zurich (associated with VME meetings) Host: Shlomo Pri-Tal, 602-438-3168

November 15   Hewlett Packard (same location as August 26)

December 13   Motorola (Phoenix, Arizona) Host: Shlomo Pri-Tal, 602-438-3168

January 10, 1989 Silicon Valley (Santa Clara, California) area

February 14   Texas Instruments (Dallas, Texas) Host: Jay Cantrell
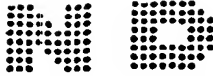
NORSK DATA   REPORT   November 1985

This report is a result of discussion among several designers at Norsk
Data.. In this report, we propose a basis for solution to many aspects
of SCI operation. This report includes :

- A Proposal for SCI Operation by Knut Alnes

    - Id codes
    - Ring to ring addressing
    - Ring arbitration
    - SCI reset
    - Acknowledge and response use
    - Packet formats

- A Proposal for CRC Error Detection by Ernst H. Kristiansen

- A Proposal for TLB Handling by Bjorn Bakka

SCI : A Proposal for SCI Operation

Norsk Data A/S, Oslo, Norway
November 10, 1988
by Knut Alnes

---

1. SCI Id Codes

A SCI address is defined as follows :

63	47	0
Node id	Address offset	
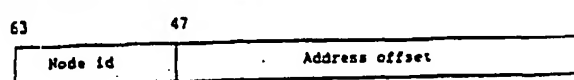
The node id is a 16 bit unique code for each node in a SCI system. The node id can be divided into the following format :

15	5	0
Global Id	Local Id	
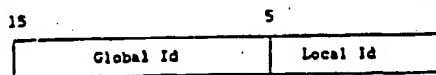
The 10 MSB of the node id is the global id code. The global id code is used only when global transactions are made. The local id is the 6 LSB of the node id. The local id is used on local transactions. for instance on a local ring.

A node ONLY looks at the local id to determine if the packet is for him. In a SCI system with several rings. the global id acts like a ring number which informs a switch which way to route a packet.

2. Ring to Ring Addressing

In a SCI network consisting of several rings. a ring switch must be able to determine which ring should receive the packet. Located in the packet header is information which tells the switch that the packet should be picked up. A switch is thus addressed not by a node id. but by information located in the header (see Packet Header section). When a switch picks up a packet. it looks at the global id code (10 MSB of node id) to determine which ring should receive the packet. The global id code will be used as a ring number in systems consisting of several rings. In a crossbar type of network. the global id code can be used to access different sections of the network.

3. SCI Ring Arbitration

In a ring, a node may be prevented from transmitting by other nodes. An arbitration mechanism must be developed to assure that each node in a ring has fair access to the network. The following proposal ensures a round robin arbitration where each node has fair access to the ring.

Use a Go bit, located in the node word of the packet header, to disable or enable other nodes' use of the network. Use the following algorithm :

1. When a master transmits packet A, the Go bit in packet A is reset to 0.

2. If another node in the ring wants to transmit, that node sets the Go bit in packet A to 1.

3. When the original master receives packet A (the acknowledge) with the Go bit set, that master is disabled from transmitting.

4. A disabled master may transmit when either :

   1. It has seen a packet with the Go bit reset to 0, and it is receiving idles.

   2. It has not seen a packet with the Go bit reset to 0, but an excessive amount of idles (more than the number of words in the largest packet) is seen.
   This indicates that the node which set a Go bit somehow did not send a packet.

This algorithm requires that a master must receive an acknowledge such that it can be disabled if any other nodes in a ring are prevented from transmitting. In a multi-ring network, switches must also acknowledge packet reception. This means that switches also must be enabled and disabled from transmitting just like the nodes. Hence, both nodes and switches use the ring arbitration algorithm.

4. SCI Interface Reset.

During SCI reset, the following must be done :

1. Assign node identifications.
2. Initialize registers.

We have considered several alternatives to reset the SCI network. All alternatives include the use of a node defined as a master. Each local ring must have a master and in a crossbar network there must also be a master. Since we must have a garbage packet collector, it is natural to assign the master function to that node. On a local ring, we call the master the ring master.

Assignment of node identifications can be made by software or hardware. Hardware assignment using geographical address is very simple, but may not be desirable. In the following section, we propose

a protocol to assign node identifications by packet sending.

4.1 Software Assignment of Node Id's.

At reset, the nodes in the SCI network do not have a node id. In order to access the nodes, a special software startup sequence must be developed. During software assignment of local and global id codes, the following must be done :

1. Activate the ring master.
2. After the ring master is activated, it sends a Start Up packet. The start up packet contains a temporary id code for each node. The ring master uses the temporary id code to access the nodes.
3. The ring master accesses the nodes using the temporary id codes and assigns local and global id codes.

Here is more detail :

1. At reset, the ring master must be activated. The ring master can activate itself or it can be started by receiving start packets from the other nodes in the ring.

2. For the start up sequence, we must define one new transaction. The transaction must be able to traverse a network such that it reaches every node once and only once. On a local ring this is no problem. However, in a full crossbar type of network, we must define a route such that each node is reached only once (this can be done because a crossbar network can be traversed as a ring). We can call this transaction the Start Up transaction. The Start Up transaction informs a node to read a counter, included in the packet, and accept the counter contents as its temporary id code. The packet is then forwarded to the neighbor node according to the above described route. The temporary id code is not the local id code nor the global id code. The purpose of the Start Up transaction is to assign temporary id codes such that the ring master can access a node and write local and global id codes.

   The temporary node id assignment can be done as follows : The master sends the startup packet to its neighbor with a counter set to 1. The neighbor node sets its temporary node id equal to the counter contents. It then increments the counter and sends the packet to its neighbor. No acknowledge is sent to the master. The packet will travel around the ring once and the counter will be incremented each time it passes a node. When the master receives the packet, the counter will have a value equal to the number of nodes in the ring which responded. The Start Up packet is explained in detail in the Start Up Packet section.

3. The master can now access an individual node using the temporary id codes. It will send a new packet to each node to reassign the temporary node id's to a 16 bit id code. When the packet is received by each node, the packet is acknowledged and sent back to the master.

4.2 Register Initialization.

After the node identifications are done. the master will initialize
the SCI interface registers. This is done by sending packets which
write into the register address space.

5. SCI Register Space

Each node connected to a SCI network must have a set of registers
which can be read and written into. The amount of register space
allocated is currently 4 kb on each node. The registers can be
addressed as follows :

1. Use the Node Id code to access the node.

2. A special command in the Command field of the packet header
   specifies a register operation.

3. The 12 lower bits of the address part of the packet is used to
   access an individual register.

4. The other bits of the address can be used to inform what type of
   register operation which is to be performed.

```
 63         47         xx        11                    0
 ┌─────────┬──────────┬─────────┬──────────────────────┐
 │ Node Id │          │Operation│ Register address      │
 └─────────┴──────────┴─────────┴──────────────────────┘
                                    │
                                    └── Used to access register
                               │
                               └── Register operation
       │
       └── Node to which the
           register operation
           apply
```
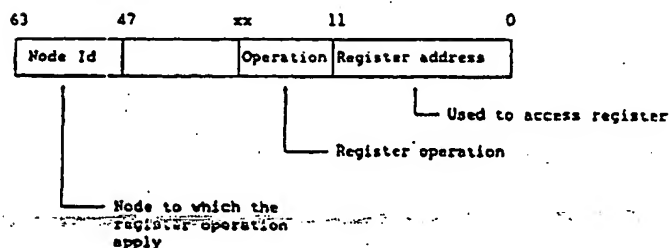
6. Cyclic Redundancy Code.

A CRC provides good error detection and we propose that only CRC is used for error detection in a packet. The CRC is computed for each word in the packet and the CRC code for the whole packet is attached at the end of the packet as shown.

```
+-------------------+
|                   |
|      Header       |
|                   |
+-------------------+
|                   |
|      Address      |
|         &         |
|       Data        |
|                   |
+-------------------+
|       CRC         |
+-------------------+
```
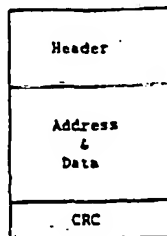
When a node receives a packet, it can compute the CRC as the packet is being received. After the whole packet is received, the computed CRC is compared with the CRC code at the end of the packet. If they match, the packet is error free. For more detail, see the CRC report by Ernst R. Kristiansen.

7. Use of Acknowledge

The need for and the use of acknowledge has raised much discussion. Our conclusion is as follows :

We need acknowledge in the following situations :

1. Ack by slave on read request.
2. Ack by master on read response.
3. Ack by slave on write request.
4. Ack by master on write response.

We need acknowledge :

1. In order for the ring arbitration protocol to provide fair access to the network.
2. Fast arbitration.
3. Simple and correct operation of split response. The implementation will be easier if a master receives an acknowledge before a new request is be issued.

Our proposal for use of acknowledge is based on the following :

1. A local target and local source field is located in the node word of the packet header. (see Packet Header section)
2. Use of CRC code.
3. A node ONLY looks at the local target field to determine if the

packet is for him.
4. Strip off address and/or data.

When a master transmits a packet, the node word of the header contains two id code fields. The local target field (16 LSB of node id) contains the local id code of the slave. The local source field contains the local id code of the transmitting master. At the end of the packet, the master attaches the computed CRC code.

A slave ONLY looks at the local target id code to determine if he should pick up the packet. If the local target id is equal to the local id code of the slave, then the packet will be received. The slave will then swap the local target and local source fields in the packet header. When the packet is returned to the master, it will pick up the packet beause the local target field matches its local id code. Before the slave sends the packet back to the master, it must compute the CRC for the whole paket. The header will be delayed until the CRC is computed and compared with the CRC code at the end of the packet. If they match, the CRC code is attached to the end of the header and the packet is sent back to the master. Thus, the address and/or data part is stripped off. If the CRC codes did not match, the slave must notify the master that something went wrong. The slave must force the master to retry the packet. This can be done in a clever way by returning a CRC code to the master which will force it to retry the packet. Simply invert some of the bits in the CRC code which is returned to the master, and the master will compute a CRC error and retry the packet.

When the master receives the packet, it looks at the local target to determine if the packet should be picked up. If the packet is for him, the master looks at the ACK bit to determine if an acknowledge packet was received. If the computed CRC matches the received CRC, the acknowledge is received correctly. If the comparison indicates an error, the original request is retried.

8. Busy Retry

When a node receives a packet, but the node is busy, the local target and local source fields are swapped and the busy bit is set. The packet is then transmitted back to the sender. When the original master picks up a packet with the busy bit set, the local target and source are swapped and the packet is transmitted again. After a certain busy retry count, the original master loggs an error and the packet is removed from the network.

The busy retry mechanism is applied to request, response, and acknowledge transactions. The benefit of swapping the local id codes is that the sender always has control over the busy retry. The sender may delay the retry if it detects that a slave is very busy.

The packet is divided into a header, address, data and error check part. These parts are discussed in the next sections.
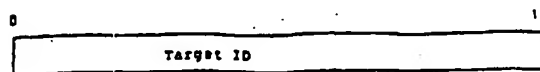
```
┌─────────────────┐
│                 │
│     Header      │
│                 │
├─────────────────┤
│                 │
│    Address      │
│       &         │
│     Data        │
│                 │
├─────────────────┤
│                 │
│   Error check   │
└─────────────────┘
```
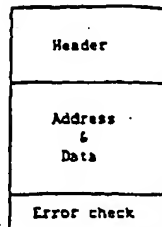
9.1 Packet Header

The packet header can be divided into four fields as shown below. The node word field contains information needed for packet routing and packet identification on a local network. The target and source word fields are used for routing on global accesses, and the command field specifies the SCI network transactions.

```
┌─────────────┐
│  Node Word  │
├─────────────┤
│ Target Word │
├─────────────┤
│ Source Word │
├─────────────┤
│  Command    │
└─────────────┘
```
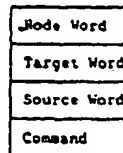
9.1.1 Node Word Field

The contents of the node word field is essential if we want to minimize the delay through a node. The following affects the delay :

1. Time to recognize the target field and start receiving the packet.
2. Time to determine acknowledge and response status.
3. Time to switch local target and source fields.

Our goal is to include as much information into the node word as possible. We feel that the following MUST be included :

1. Local target.
2. Local source.
3. Start up information.

The local target and local source must be included because it makes
node identification faster. Also, it makes it faster to swap the
target and source fields on acknowledge and response.

In addition, start up information should be included in the header
such that a node can identify a start up packet by
looking at the node word field.

Here is our proposal of what a node word field may look like :
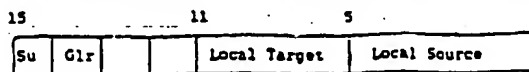
| 15 | | 11 | | 5 | | 0 |

| Su | Glr | | | Local Target | Local Source |

Su   Start up bit. This bit tells the node that the packet contains
        a counter which contents will be the node's temporary id code.

Glr   The global ring bit is used to inform a ring switch if the
        packet should be forwarded in the local ring or if the packet
        should be switched over to a global ring. Used only on global
        transactions. (see Ring to Ring Addressing section)

Local   The Local target field contains the local target id. The local
Target target id is the 6 LSB of the node id.

Local   The Local source field contains the local source id. The local
Source source id is the 6 LSB of the node id.

If we include the local target and local source in the node word, we
only need to put the 10 MSB of the node id in the next two fields. The
advantage of this is that by using only 10 bits for the global target
and source, we make room for 12 additional bits in the target
word and source word fields.

9.1.2 Target Word Field.

The global target id field contains the 10 MSB of the node id. The
additional bits are used for other header information. On global
accesses the global target and global source fields may be switched on
an acknowledge and response.

| 15 | | | | 9 | | | 0 |

| Bsy | Go | Old | Ack | Gla | | | Global Target |
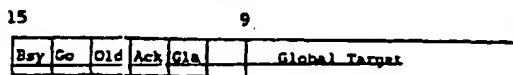
Bsy   The busy bit is set if a node is busy (slave fifo is full).

Go   The go bit is used during ring arbitration.

Old   The old bit is set by the ring master.

Ack   The Ack bit is set if a slave has received a packet.

Gla   Global access bit. This bit is set by a server when the

acknowledge is sent to the client. The Gla bit informs a switch to forward the packet to a global ring. Used only on global transactions.

9.1.3 Source Word Field

The global source id field contains the 10 MSB of the node id. The additional bits can be used for other header information. On global accesses the global target and global source fields may be switched on an acknowledge and response.

```
15.                        9                    . 0
 ┌────┬────┬────┬────┬────┬────┬──────────────────┐
 │Sci.│Sci │Sci │Sci │Sci │Sci │  Global Source   │
 └────┴────┴────┴────┴────┴────┴──────────────────┘
```
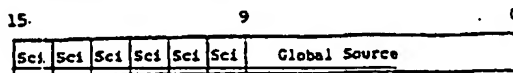
Sci.   SCI network bits. These bits can be used by the specific SCI
          network implementation. Switches may use these bits· depending
          on the network configuration and the operation of the
          switches. Here are some proposed uses :

          Switch id. We may need to address switches by providing a
          switch id. The reason for this is that a switch may need to
          remove a packet which it sent out, thus it must recognize its
          own packet. Also, other switches may be prevented from picking
          up packets. For ring arbitration, the correct switch must
          remove the acknowledge which is returned from a node or
          another switch.

          Busy retry. A switch may need to retry a packet a certain
          amount of times. The network bits can be used to hold the
          retry counter of a packet.

9.2 Command Field

The command field contains the command from the client to the server.

9.3 Address Field

The address fields contains the 48 bit address within a node.

SCI-10Nov88-doc23-010

9.4 Data Field

The data fields contains the data block to be transferred.

9.5 CRC Field.

The CRC field contains the Cyclic Redundancy Code.

9.6 Start Up Packet for Software Id Assignment.

Here is a proposed format for the Start Up packet which assigns temporary id codes :

| Node Word |
| Don't care |
| Don't care |
| Don't care |
| CRC |

The extra don't care fields are inserted to make all headers the same length. The contents of the node word field is :

```
15      12      11                                    0
|  ..  | Su=1 | Local Target=1 | Local Source = 0.1.2..n |
```

At reset, all nodes have node id set to 1. The ring master has id code 0. The ring master sends the start up packet to its neighbor node. The neighbor node accepts the packet because its id is 1 and the local target id is 1. Because the Su bit is set, the node increments the local source field and its temporary node id becomes the new contents of the local source field. The start up packet is now sent to the next neighbor and the same operation is repeated. When the packet has travelled around the ring, the local source field contains the number of nodes in the ring.

E.D.VA. CA 00-524
I 206692

863 FH PG 0809

10. SCI Packet Usage

1. A read request packet contains a header and an address part. A CRC code is attached to the end of the packet. When the slave acknowledges the packet, the address part is stripped off and the CRC code is attached after the header.

REQ

Node Word
Target Word
Source Word
Command
A(47:32)
A(31:16)
A(15:0)
CRC

ACK

Node Word
Target Word
Source Word
Command
CRC

2. A read response packet contains a header, address part, and data part. On a cache coherence transaction, a pointer may be returned. The acknowledge packet only returns the header with the CRC code.

RESPONSE

Node Word
Target Word
Source Word
Command
A(47:32)
A(31:16)
A(15:0)
Ptr
Data n

Data 0
CRC

ACK

Node Word
Target Word
Source Word
Command
CRC

E.D.VA. CA 00-524
I 206693

3. Write request packets include a header, address part, and a data part. The acknowledge packet only returns the header. The acknowledge packet is returned to maintain fair and fast arbitration.

```
        REQ
   ┌──────────────┐
   │  Node Word   │
   ├──────────────┤
   │  Target Word │
   ├──────────────┤
   │  Source Word │
   ├──────────────┤
   │  Command     │
   ├──────────────┤                      ACK
   │  A(47:32)    │              ┌──────────────┐
   ├──────────────┤              │  Node Word   │
   │  A(31:16)    │              ├──────────────┤
   ├──────────────┤              │  Target Word │
   │  A(15:0)     │              ├──────────────┤
   ├──────────────┤              │  Source Word │
   │  Data n      │              ├──────────────┤
   └──────────────┘              │  Command     │
                                 ├──────────────┤
   ┌──────────────┐              │  CRC         │
   │  Data 0      │              └──────────────┘
   ├──────────────┤
   │  CRC         │
   └──────────────┘
```

4. Write response packets are needed to return the status of the tag bits and a pointer to the head of the sharing list. This is necessary during uncached write transactions.

```
      RESPONSE
   ┌──────────────┐
   │  Node Word   │
   ├──────────────┤
   │  Target Word │
   ├──────────────┤
   │  Source Word │
   ├──────────────┤                      ACK
   │  Command     │              ┌──────────────┐
   ├──────────────┤              │  Node Word   │
   │  A(47:32)    │              ├──────────────┤
   ├──────────────┤              │  Target Word │
   │  A(31:16)    │              ├──────────────┤
   ├──────────────┤              │  Source Word │
   │  A(15:0)     │              ├──────────────┤
   ├──────────────┤              │  Command     │
   │  Ptr         │              ├──────────────┤
   ├──────────────┤              │  CRC         │
   │  CRC         │              └──────────────┘
   └──────────────┘
```

Μ

N D NORSK DATA   REPORT   January 1989

This report includes :

● A Proposal for SCI Operation by Knut Alnes

    ● Packet format

    ● Priority arbitration

    ● Use of acknowledge

    ● Busy retry

    ● Fault retry

    ● Global SCI operations

● Logical Level Proposals by Bjørn Bakka

    ● Broadcast Update

    ● Broadcast Invalidate

    ● Packet reject

    ● Sequencing

SCI : A Proposal for SCI Operation

Norsk Data A/S. Oslo, Norway
January 6. 1989
by Knut Alnes

1. Packet Format

A packet is divided into a header, address, data and error check part.
These parts are discussed in the next sections.

```
┌─────────────┐
│             │
│   Header    │
│             │
├─────────────┤
│             │
│   Address   │
│      &      │
│    Data     │
│             │
├─────────────┤
│     CRC     │
└─────────────┘
```
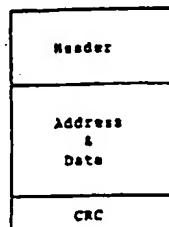
1.1 Packet Header

The packet header can be divided into four fields as shown below. The
target word field contains information needed for packet routing and
packet identification. The Source Word field contains the ID code of
the sender. The Flow Control word contains information used to control
the flow of packets. The command field specifies the SCI network
transactions.

```
┌──────────────┐
│ Target Word  │
├──────────────┤
│ Source Word  │
├──────────────┤
│ Flow Control │
├──────────────┤
│ Command      │
└──────────────┘
```
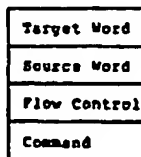
1.1.1 Target Word Field

The target word field contains the 16 bit identification code for the target of the transaction.

```
0                                              15
┌────────────────────────────────────────────────┐
│                  Target ID                       │
└────────────────────────────────────────────────┘
```

1.1.2 Source Word Field

The Source Word field contains the source id of the sender.

```
0                                              15
┌────────────────────────────────────────────────┐
│                  Source ID                       │
└────────────────────────────────────────────────┘
```
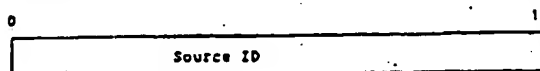
1.1.3 Flow Control Field

The Flow Control field contains information needed to control the flow of packets in a SCI network.

```
0                                                          15
┌────┬────┬────┬────┬────┬────┬────┬────┬────┬────┬────┬────┬────┬──────┐
│Ack │Bsy │Old │Tp0 │Tp1 │Co1 │Co2 │Co3 │Co4 │Ra │Rp │Su │Pd │      │
└────┴────┴────┴────┴────┴────┴────┴────┴────┴────┴────┴────┴────┴──────┘
```
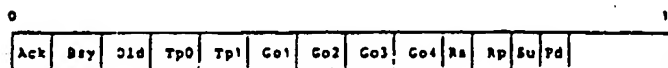
Ack
The Ack bit is set by the target when a packet is received error free.

Bsy
The Bsy bit is set by the target if the node cannot accept the packet because its queues are full.

Old
The Old bit is set by a ring cleaner to prevent endless circling of packets.

Tp1-Tp0
Tp1-Tp0 contains the coded transaction priority. We assume four priority levels. Tp1-Tp0 are set by the original server and are not manipulated by other nodes.

Co4-Co1
Co4-Co1 are the priority Co bits used during arbitration. If the Co4 bit is set, then the transaction has the highest priority. If the Co1 bit is set and Co4-Co2 are reset, then the transaction has the lowest priority. We use four Co bits instead of 2 decoded bits to make the manipulation of the Co-bits easier and faster.

E.D.VA. CA 00-52
I 206764

Ra
If the Ra bit is 1, then packet Reject is Allowed. This means that the packet is put into a slave's processing queue but may later be rejected from the queue and the transaction must be retried by the

master. If the Ra bit is 0, then packet Reject is not allowed (See
Bjorn Bakka's report for more detail). The Ra bit is set or reset by
the master on a request.

Rp
The Received&Processing bit is used together with the Ra bit. For
transaction where sequencing is crucial, the request packet contains
Ra=0 meaning reject is not allowed. The Rp bit is set or reset by the
slave to inform whether the packet may or may not be rejected. If the
Rp bit is 1 in the acknowledge packet, then the request will not be
rejected and the master may issue a new request. If the Rp bit is 0,
then the request may be rejected and the master must wait for the
response before a new request may be issued. (See Bjorn Bakka's
report for more detail)

Su
Start up bit. This bit tells the node that the packet contains a
counter which contents will be the node's temporary id code.

Pd
The packet delete bit is used during transactions accross several
rings and switch networks. This bit informs the slave to delete the
packet from the network. See the Global SCI Operations section for
detailed use.

1.1.4 Command Field

The command field contains the command from the client to the server.
Also, the command field contains an identifier (previously called
sequence number) which is used by the master to identify the response
with the corresponding request.

0   15

Identifier	Command
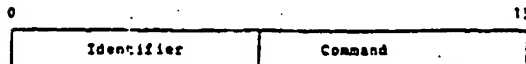
1.2 Address Field

The address fields contains the 48 bit address within a node.

1.3 Data Field

The data fields contains the data block to be transferred.

1.4 CRC Field

The CRC field contains the Cyclic Redundancy Code. See report by Ernst M. Kristiansen.

2. SCI Arbitration

In this section, we discuss two arbitration implementations. The first implementation is the round robin arbitration which we proposed earlier. The second implementation is an direct access implementation which removes the arbitration problem, but puts severe limitations on the network protocol. This implementation is mentioned because of its simplicity and in order to have two arbitration implementations which can be compared. The comparison will focus on arbitration time and transfer time in a ring network. After this discussion, we present simulation results for the round robin algorithm.

2.1 Round Robin Arbitration

In a ring, a node may be prevented from transmitting by other nodes. An arbitration mechanism must be developed to assure that each node in a ring has fair access to the network. The following proposal ensures a round robin arbitration where each node has fair access to a ring.

Use a Go bit, located in the flow control word of the packet header, to disable or enable other nodes' use of the network. Use the following algorithm :

1. When a master transmits packet A, the Go bit in packet A is reset to 0.

2. If another node in the ring wants to transmit, that node sets the Go bit in packet A to 1.

3. When the original master receives packet A (the acknowledge) with the Go bit set, that master is disabled from transmitting.

4. A disabled master may transmit when either :

   1. It has seen a packet with the Go bit reset to 0, and it is receiving idles.

   2. It has not seen a packet with the Go bit reset to 0, but an excessive amount of idles (more than the number of words in the largest packet) is seen.
      This indicates that the node which set a Go bit somehow did not send a packet.

This algorithm requires that a master must receive an acknowledge such that it can be disabled if any other nodes in a ring are prevented from transmitting. In a multi-ring network, switches must also acknowledge packet reception. This means that switches also must be enabled and disabled from transmitting just like the nodes. Hence, both nodes and switches use the ring arbitration algorithm.

E.D.VA. CA 00-524
I 206766

2.3 Direct Access Arbitration

This implementation puts two restrictions on the protocol. First, only one pending acknowledge is allowed. Second, packet stripping is not allowed. The advantage of this implementation is that the arbitration time is always zero, thus a node has always direct access to the ring.

The algorithm is as follows :

1. Start transmitting when the bypass fifo is empty. If a packet is entering the node interface, the packet is buffered in the bypass fifo.

2. When the node receives the acknowledge, the bypass fifo is emptied while the acknowledge packet is accepted into the slave fifo. This leaves an empty bypass fifo after the transaction is completed. When a node is waiting for an acknowledge, the bypass fifo can be empty or filled. However, after the acknowledge is received, the bypass fifo will ALWAYS be empty thus allowing the node to transmit another packet.

As mentioned, this implementation may not be useful because of the restrictions it puts on the network protocol.

2.3 Comparison between the implementations

There is a trade off between arbitration time to get access to the ring, and the transfer time once the packet is sent onto the ring. With zero arbitration time, which is the case for direct arbitration, the chance of filling up the bypass fifos is high. Since the bypass fifos are likely to be full, the transfer time is proportional to the number of filled bypass fifos.

With higher arbitration time (which is the case for round robin) the transfer time will be lower because very few packets (most likely one or two for a small ring) will be in the ring at any time, and the bypass fifos will most likely be empty. This is true for round robin arbitration. If the ring traffic is consistently high and all nodes want to transmit. If the ring has been "quiet" for a while, and suddenly several accesses are done at once, the arbitration time is low (zero) but the transfer time is likely to be higher because the bypass fifos will be filled.

Below is an analysis of worst case arbitration and transfer times for the two arbitration implementations. The analysis is based on the following restrictions :

- Ring implementation
- Fixed packet length
- No busy packets
- No stripping of packets

If we take into account varying packet lenghts, busy retry and packet stripping, the analysis becomes more complex. However, for a worst case discussion the analysis should be useful.

3. Direct Access.

Worst case transfer time

    Twc = (N-2) * node delay * 2ns

where N = number of nodes in the ring and the node delay is the delay through the bypass fifo.

Ex. N= 10 nodes, node delay with bypass fifos full is 40 (80 bytes packet length) + 2 input register delays

    Twc = 8*42*2ns = 672ns

Worst case arbitration time

    Awc = 0ns

Total time = Twc + Awc = 0 + 672 = 672ns.

2. Round Robin.

If we use a round robin algorithm the following worst case (maximum use of the ring, all nodes want to send) situations can occur :

Worst case transfer time

    Twc = (N-2) * node delay * 2ns

Ex. N=10, node delay with bypass fifos empty (2 input register delays)

    Twc = 8*2*2ns = 32ns

Worst case arbitration time

    Awc = (N-1) * packet length * 2ns

Ex. N=10, packet length is 40 (80 bytes)

    Awc = 9*40*2 = 720ns

Total time = Twc + Awc = 32 + 720 = 752ns.

Worst case average arbitration time

    Assuming packet length is 40 words (80 bytes) and node delay is 40*2ns = 80ns. N is the number of nodes. Take all request possibilities and divide by the number of requests.

    At = 80*(N-1) + 80(N-2) + ... + 80(0)/N
       = 80*(N*N - N(N+1)/2)/N
       = 80*(N-1)/2

    At = packet length * 2ns * (N-1) / 2

Ex. N=10, packet length is 40 (80 bytes)

At = 40*2ns*9/2 = 360ns.

As seen from the above comparison, the round robin arbitration may not perform all that bad even if the arbitration time may seem high. The

advantage of this algorithm is that it does not restrict the protocol
and that it can be applied to a priority scheme as well. The
performance of a round robin arbitration still needs to be verified.
Although we have done simulations on the actual behavior of the
algorithm we lack testpatterns which shows realistic activity on a SCI
network.

2.4 Round Robin Implementation

During simulation of the proposed round robin algorithm, we discovered
a potential problem. This problem was related to the setting of the Go
bit when a node wants to transmit. The problem occured when :

1. No packets are in the ring.

2. Many nodes send a request onto the ring at the same time.

2. Each requesting node sets the Go bit because it has more packets
   to send.

3. The Go bit is set before the node has received an acknowledge on
   the request.

If no packets are in the ring and several nodes start to transmit at
once, all nodes stop transmitting after receiving the acknowledge with
the Go bit set. Since all nodes become disabled from transmitting, no
nodes attempt to transmit another packet. After waiting for a certain
number of idle cycles (more than the number of words in the largest
packet) all the nodes start to transmit again, and the same procedure
repeats. For a simultaneous ring accesses of this type the network
usage will be low because many nodes start to transmit and then wait
for a while before they transmit again.

By trying different strategies for setting of the Go bit, we
discovered that the problem could be solved by waiting for the
acknowledge on the request before the Go bit could be set. This causes
all the nodes to be enabled for transmission during simultaneous
accesses and improves the network usage. The following code gives more
detail :

```
if Want_To_Transmit & Incoming_Packet & No_Pending_Ack then
  Set Go bit
  Node is enabled for transmission
else if Ack_Or_Response_Packet & Go_Bit_Set then
  Node is disabled from transmitting
else if Idle_Count = Max then
  Node is enabled for transmission
```

2.5 Priority Arbitration

We base our priority arbitration proposal on the previously discussed round robin arbitration using a Go bit. Instead of using one Go bit, we now extend our algorithm to four Go bits, one for each priority level. Transactions which are on the same priority level will follow the round robin arbitration. The flow control word of the header will contain four Go bits Go4-Go1. These bits can be manipulated by any node while packets pass through a node interface. In addition, two transaction priority bits Tp1-Tp0 are used to tag the packet with the original priority. This priority is set by the original server only. The following priority levels can be used:

Priority level	Go4	Go3	Go2	Go1
4 Critical	1	0	0	0
3 Real time	0	1	0	0
2 Foreground	0	0	1	0
1 Background	0	0	0	1

The four priority levels are coded in four bits for easy and fast manipulation. These bits will be set and reset as packets pass a node interface. We may not have time to decode priority levels and compare priorities so we save the incoming Go bits and set the outgoing Go bits while a packet passes by. The saved Go bits are then decoded and priorities are compared after the header has passed the node interface.

The priority arbitration requires the nodes to code their priority levels into the packet header using the Go bits. Nodes which sees packets with priority levels higher than their priority will be disabled from transmitting. Nodes with the same priority levels will follow a round robin arbitration. When a node with a high priority wants to transmit, it codes its high priority into the first packet it sees. The node accepting this packet will send out a small packet which travels around the ring once. This packet disables all nodes with lower priority. This priority increase packet is sent to assure fast arbitration for the high priority node.

1. When a node sends a packet, it codes its priority into the Go bits. This is done by setting all Go bits lower than its own priority. If the priority is 3 then the Go4-Go1 = 0011. The Go bits for priority level 1 and 2 are set to disable nodes with these lower priorities. Go3 is 0 to enable another node according to the round robin mechanism. This ensures round robin on the same priority level.

   ```
   if Enabled & Incoming_Idle then
     Start sending
     for I := 1 to Priority_Level-1 do
       Go[I] := 1
   ```

2. If a node is disabled and wants to send, then it sets the Go bit corresponding to the priority level of the node and all lower priority levels. If the incoming Go bits are 0011 and the node has priority 4, then the outgoing Go bits are 1111.

   ```
   if Disabled & Wants_To_Send & Incoming_Packet then
     for I := 1 to Priority_Level do
       Go[I] := 1
   ```
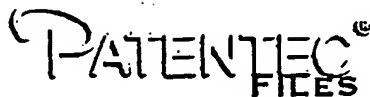
PATENTEC®
FILES

MISSING PAGE(S) FROM THE
U.S. PATENT OFFICE
OFFICIAL FILE WRAPPER

Page 9

Our proposal for use of acknowledge is based on the following :

1. A target and source field are located in the packet header. (see Packet Header section)

2. Use of CRC code.

3. A node ONLY looks at the target field to determine if the packet is for him.

4. Strip off address and/or data.

When a master transmits a packet, the packet header contains two id code fields. The target field contains the id code of the slave. The source field contains the id code of the transmitting master. At the end of the packet, the master attaches the computed CRC code.

A slave ONLY looks at the target id code to determine if he should pick up the packet. If the target id is equal to the id code of the slave, then the packet will be received. The slave will then swap the target and source fields in the packet header. When the packet is returned to the master, it will pick up the packet beause the target field matches its id code.

Before the slave sends the packet back to the master, it must compute the CRC for the whole paket. The header will be delayed until the CRC is computed and compared with the CRC code at the end of the packet. If they match, the CRC code is attached to the end of the header and the packet is sent back to the master. Thus, the address and/or data part is stripped off. If the CRC codes did not match then parts of the packet were damaged during transmission. Since the target word or source word may be damaged, the packet should be removed from the network. This means that the original client will get a time-out and must retry the request.

When the master receives a packet, it looks at the target to determine if the packet should be picked up. If the packet is for him, the master looks at the ACK bit to determine if an acknowledge packet was received. If the computed CRC matches the received CRC, the acknowledge is received correctly. If the comparison indicates an error, the original request is retried.

4. SCI Packet Usage

1. A read request packet contains a header and an address part. A CRC code is attached to the end of the packet. When the slave acknowledges the packet, the address part is stripped off and the CRC code is attached after the header.

REQ

Target
Source
Flow Control
Command
A(0:15)
A(16:31)
A(32:47)
CRC

ACK

Target
Source
Flow Control
Command
CRC

2. A read response packet contains a header, address part, and data part. On a cache coherence transaction, a pointer may be returned. The acknowledge packet only returns the header with the CRC code.

RESPONSE

Target
Source
Flow Control
Command
A(0:15)
A(16:31)
A(32:47)
Ptr
Data n

Data 0
CRC

ACK

Target
Source
Flow Control
Command
CRC

E.D.VA. CA 00-524
! 206773

SC1-6Jan89-doc31-p12

3. Write request packets include a header, address part, and a data part. The acknowledge packet only returns the header. The acknowledge packet is returned to maintain fair and fast arbitration.

REQ

Target
Source
Flow Control
Command
A(0:15)
A(16:31)
A(32:47)
Data n

Data 0
CRC

ACK

Target
Source
Flow Control
Command
CRC

4. Write response packets are needed to return the status of the tag bits and a pointer to the head of the sharing list. This is necessary during uncached write transactions.

RESPONSE

Target
Source
Flow Control
Command
A(0:15)
A(16:31)
A(32:47)
Ptr
CRC

ACK

Target
Source
Flow Control
Command
CRC

E.D.VA. CA 00-524
I 206774

5. Busy Retry

When a node receives a packet, but the node is busy, the target and source fields are swapped and the busy bit is set. The packet is then transmitted back to the sender. When the original master sees a packet for him with the busy bit set, it has three possible options to follow :

1. The master can swap the target and source and send the packet through the bypass fifo. This is an immediate retry which effectively gives busy packets the highest priority.

2. If the slave fifo is empty, the master may accept the busied packet into the slave fifo. The master can then arbitrate and transmit the contents of the slave fifo when it is granted access to the network.

3. The master can remove the busied packet from the network. A copy of the request will be put into the master fifo and the master will arbitrate for access to the network. When access is granted, the master fifo contents is transmitted and the busied request is retried.

We propose the use of option 3. Option 3 provides fair arbitration because the master must arbitrate before it can retry a request. Also, this option is independent of the slave fifo being empty or not. Since a master must save a copy of a request until it receives an acknowledge or a response, it can use this copy to retry a request.

The busy retry mechanism is applied to request, response, and acknowledge transactions. The benefit of swapping the id codes is that the sender always has control over the busy retry. The sender may delay the retry if it detects that a slave is very busy. Also, swapping of target and source makes the packet identification easier and faster.

6. Fault retry

Fault retry has raised much concern due to the complexity of ensuring that we do not cause sequencing errors. Some instructions must be processed in a certain sequence and it is essential that this sequence is not disturbed by retried transactions. We propose a fault retry strategy based on the following:

1. The master is responsible for retrying requests.

2. The slave is responsible for retrying responses.

3. The master is responsible for issuing requests in the correct order. This means that a master may need to wait for an acknowledge and even a response on a request before it can issue a new request. However, these restrictions only apply to transactions which are sensitive to sequencing errors. For a detailed discussion on sequencing problems, see Bjorn Bakka's special report.

4. The slave must ensure that it does not execute the same instruction twice due to retried requests. This is not true for all instructions, but when sequenced instructions are involved this must be guaranteed.

One way to screen retried requests is to save a copy of all requests which have been executed. When an instruction is executed, a copy of the request is added to a list of processed requests. When a request is retried, it must be checked against the list to see if it has already been processed. This comparison must be made by id code and sequence number. If the retried request is found in the list, then the retried request is ignored. Otherwise the request is processed. Request can be deleted from the processed list when an acknowledge on the corresponding response has been received.

This mechanism works well with our proposed four-phase handshake (request-acknowledge, response-acknowledge). Also, the mechanism scales well as the number of pending acknowledges and responses increase.

7. Global SCI Operations

This section explains the different transactions used during global accesses across a SCI network. The transactions which are explained are the same for read, write, and cache coherence operations.

The proposed transactions are based on the following :

1. Fair arbitration must be maintained. On global accesses, an arbitration packet is returned from and to switches to ensure fair access for other nodes.

2. The client is responsible for retrying requests. The server is responsible for retrying responses.

3. Switches look at the target to determine where the packet should be routed.

The advantages of these transactions are as follows :

1. Fair access is maintained.
2. Fast packet detection by only looking at the target.
3. Fault retry safe.

The proposed transactions are shown on the following pages.

7.1 Request - acknowledge operation

The following example is based on a client node and a server node with node ids as follows:

```
CLIENT Id - 5

SERVER Id - 7
```


CLIENT 5
master
req
arbit   slave

SERVER 7
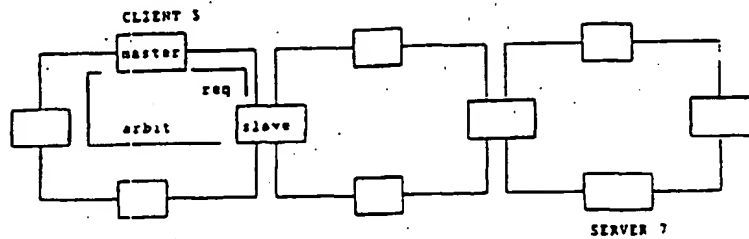
Request transaction

The three first words of the request transaction header is as follows :

Target - 7		
Source - 5		
Ack=0		PG=0

Arbit Transaction

The slave switch looks at the target and determines that it must forward the packet to the neighbor ring. The switch then swaps the target and source and sets the packet delete bit (Pd) to inform the master to pick up the packet and delete it from the ring.

Target - 5		
Source - 7		
Ack=0		Pd=1
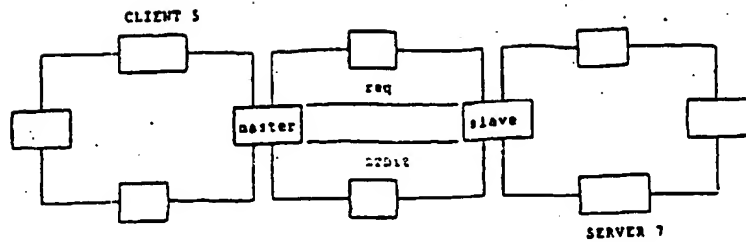
Request transaction

The master switch which received the request packet from the CLIENT will now send the packet onto the neighbor ring. The Pd bit is reset. Note that the target and source fields are not swapped.

Target - 7		
Source - 5		
Ack-0		Pd-0

Arbit transaction

The slave switch sets the Pd bit and swaps the target and source fields. The master switch picks up the packet by looking at the target and deletes the packet from the ring by looking at the Pd bit.

Target - 5		
Source - 7		
Ack-0		Pd-1
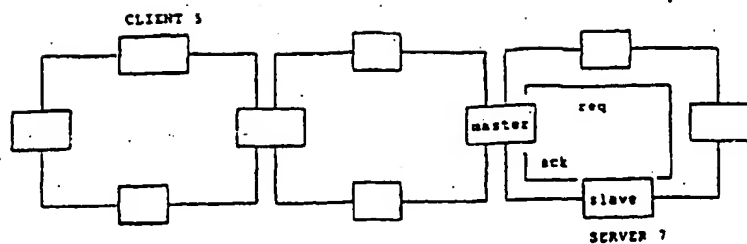
Request transaction

The master switch which received the request packet will now send the packet onto the neighbor ring. The Pd bit is reset. Note that the target and source fields are not swapped.

Target - 7		
Source - 5		
Ack-0		Pd-0

Acknowledge transaction

After the SERVER has picked up the packet, an acknowledge packet is generated. This is done by setting the Ack bit. The target and source fields are swapped and the Pd bit is set.

Target - 5		
Source - 7		
Ack-1		Pd-1

CLIENT 5

arbit

slave   master

ack

SERVER 7

Acknowledge transaction

The master switch which received the acknowledge packet from the SERVER will now send the packet onto the neighbor ring. The Pd bit is reset. Note that the target and source fields are not swapped.

Target - 5	
Source - 7	
Ack-1	Pd-0

Arbit transaction

The slave switch sets the Pd bit and swaps the target and source fields. The master switch picks up the packet by looking at the target and the Pd bit.

Target - 7		
Source - 5		
Ack-1	Pd-1	

CLIENT 5

slave

arbit

master

ack

SERVER 7

Acknowledge transaction

The master switch resets the Pd bit.

Target - 5		
Source - 7		
Ack=1		Pd=0

Arbit transaction

The CLIENT returns an arbitration packet to the master switch with the Pd bit set. Note that the CLIENT does not swap the target and source when it receives an acknowledge packet.

Target - 5		
Source - 7		
Ack=1		Pd=1

This completes the request and acknowledge operation.

7.2 Response - acknowledge operation
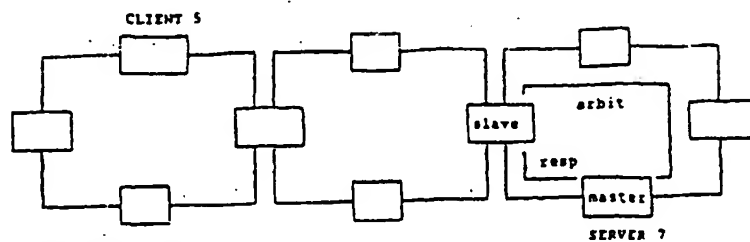
Response transaction

The SERVER generates a response packet containing data and/or status information.

Target = 5	
Source = 7	
Ack=0	Pd=0

Arbit transaction

After the slave switch has picked up the response packet, the arbitration packet is returned. The target and source fields are swapped and the Pd bit is set.

Target = 7	
Source = 5	
Ack=0	Pd=1

CLIENT 5

arbit

slave   master
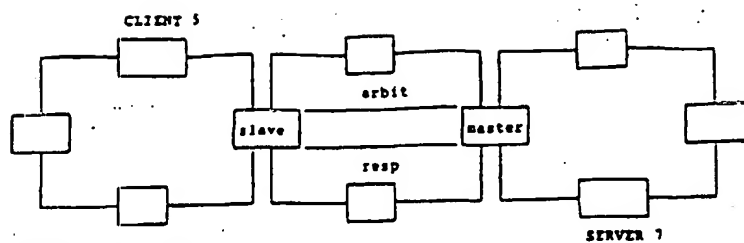
resp

SERVER 7

Response transaction

The master switch which received the response packet from the SERVER will now send the packet onto the neighbor ring. The Pd bit is reset. Note that the target and source fields are not swapped.

Target = 5	
Source = 7	
Ack=0	Pd=0

Arbit transaction

The slave switch sets the Pd bit and swaps the target and source fields. The master switch picks up the packet by looking at the target and the Pd bit.

Target = 7	
Source = 5	
Ack=0	Pd=1
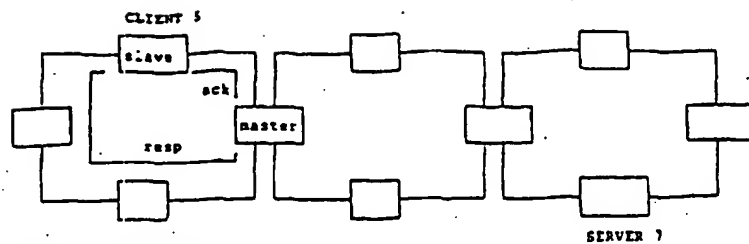
Response transaction

The CLIENT picks up the response packet. It then sets the Pd bit and does not swap the target and source.

Target = 7		
Source = 5		
Ack=0		Pd=0

Acknowledge transaction

The master switch picks up the packet and deletes it from the ring.

Target = 5		
Source = 7		
Ack=1		Pd=1
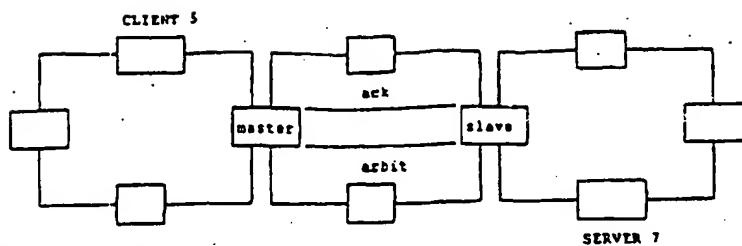
CLIENT 5

ack

master | slave

arbit

SERVER 7

Acknowledge transaction

The master switch which received the acknowledge packet from the CLIENT will now send the packet onto the neighbor ring. The Pd bit is reset. Note that the target and source fields are not swapped.

Target = 7	
Source = 5	
Ack=1	Pd=0

Arbit transaction

The slave switch sets the Pd bit and swaps the target and source fields. The master switch picks up the packet by looking at the target and the Pd bit.

Target = 5	
Source = 7	
Ack=1	Pd=1

Acknowledge transaction
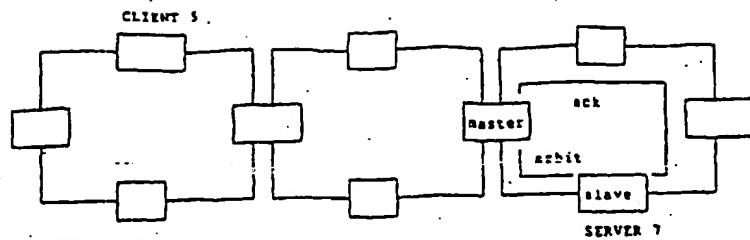
The master switch forwards the acknowledge packet onto the ring where the SERVER is located.

Target = 7		
Source = 5		
Ack=1		Pd=0

Arbit transaction

The server returns the arbitration packet to the switch master after setting the Pd bit and not swapping the target and source fields.

Target = 7		
Source = 5		
Ack=1		Pd=1

This completes the response and acknowledge operation.

SCI:
Logical Level Proposals

January 6, 1989
Bjørn O. Bakka
Ernst H. Kristiansen

Norsk Data

P.O.Box. Bogerud
N-0621 Oslo 6, Norway
Phone: +47 2 627000
FAX: +47 2 296796
Email: ehk@ndhq.uucp

1 PROPOSALS

We propose a Broadcast Update bus operation and a Broadcast Invalidate bus operation. The data item size of the Broadcast Update is 1 bit, 1 Word or any multiply of 1 Word up to 64 bytes.

To explait all advantages of these broadcast operations, we propose to introduce a new cache state: Nothingvalid. Nothingvalid is similar to Invalid by not containing a valid cache line. Sut Nothingvalid does not have a pointer to an entry with a valid cache line as Invalid do.

We propose a Reject response which is a response where the corresponding request is not processed.

We propose that the master is responsible for the sequence control, and that we use the request/acknowledge transaction as the sequence control mechanism. To be able to handle the Reject response, we propose a Reject Allowed bit in the request and a Processing bit in the acknowledge to optimize the sequence control. Without the Reject response, these bits are not needed.

2 BROADCAST OPERATIONS

2.1 PURPOSE

The purpose of Broadcast Update/Invalidate operations is to use the information already obtained during SCI-lists insertions, to decrease bus traffic and increase the ability to share time-critical data.

2.2 BACKGROUND

By using the SCI-list structure one entry is added to the list each time a cached load miss is done. Memory retains its previous state (i.e. Clean or Dirty). The new entry gets the state of the memory, while all the other entries in the list enter a Clean state.

Each time a store miss is done. the new entry enters a Dirty state. The previous head enters an Invalid state. while a Broadcast Purge is sent to the rest of the list. All following entries in the list delete themselves from the list. and the result of this is only two entries left in the list.

The only time memory is updated with valid data after it enters a Dirty state. is when a Dirty or private head deletes itself from the list. The only way a Invalid (or Uncached) entry can get hold of new and valid data. is to issue a request for the data.

The question arises whether this is the best (complexity, performance. etc.) mechanism for cached transfer, or if there are ways to improve this by making more broadcast operations available.

2.3 PROPOSAL

We propose a Broadcast Update bus operation and a Broadcast Invalidate bus operation. The data item size of the Broadcast Update is 1 bit. 1 Word or any multiply of 1 Word up to 64 bytes.

To exploit all advantages of these broadcast operations. we propose to introduce a new cache state: Nothingvalid. Nothingvalid is similar to Invalid by not containing a valid cache line. But Nothingvalid does not have a pointer to an entry with a valid cache line as Invalid do.

2.4 BEHAVIOUR

A Broadcast Update can be performed both on a Clean only list. on a Dirty/Clean/Invalid/Nothingvalid list and on a Private/Invalid/Nothingvalid list. In a Clean only list. all entries retain their Clean state during a Broadcast Update. In a Dirty list. with data item size less than 64 bytes. no states are changed. A Broadcast Update with data item size equal to 64 bytes converts any list to a Clean only list. A Broadcast Invalidate operation converts any list to a Private/Invalid/Nothingvalid list with the master as Private.

A Broadcast Update bus operation starts with a transaction from the master to memory. This transaction is acknowledged by memory. The memory controller then generates a Broadcast Update transaction to the SCI-list head. In addition to the update data item. the transaction specifies where to send the final response. The broadcast propagates the SCI-list until the end of list entry which generates a response transaction to the master.

A Broadcast Invalidate bus operation can only be performed by a master not currently part of the SCI-list. (It must first delete itself from the list.) The master starts with a transaction to memory specifying Broadcast Invalidate. The memory controller updates its pointer and state. and responds with the pointer to the previous head. The master generates a new transaction to the previous head. leaving this entry in an Invalid state. The broadcast traverses the rest of the list and leaves all entries Nothingvalid. It is in some respects similar to the Broadcast Purge operation. but the SCI-list pointers are intact.

The crucial point with Broadcast Update operations. is sequencing. It is important that when two Broadcast Updates are issued at the 'same' time. the first to access memory. is the first to access the SCI-list at all nodes. If this is not true. the operations may not leave a consistent memory image behind. The easiest way to decide 'first' and 'second' is to use memory as a syncronizing barrier and not allow any out-of-order transactions to happen.

Another interesting thing about using memory as the syncronizing barrier. is that the memory controller may check if a Broadcast Update operation actually changes data. If it does not. the memory controller may inhibit the operation. This is a particularly interesting feature for the Used and Modified bits in a Page Table Descriptor.

2.5 INVESTIGATIONS

The reason to introduce a Broadcast Invalidate. is to allow a master to grab and hold on to a cache line that is. and will be. shared by multiple nodes. While the master is working with that cache line, no one in the sharing list may easily get access to the data. When the master is finished, it updates all nodes with the new and valid data.

The use of Broadcast Invalidate should be to restrict the access to shared data that is not restricted by semaphores. That means for instance the semaphores themselves and virtual memory tables. But the question remains whether there are other and cheaper methods to obtain the same functionality?

In the Broadcast Update bus operations. there are many ways to either sentralize or desentralize the responsibility of leaving a consistent memory image behind. It is possible to desentralize everything to the memory controller which immediately issues a response to the master. Or it is possible to sentralize everything to the master which issue point-to-point accesses.

In the Broadcast Update and Invalidate Scheme. it is possible to include a Delete option. However. this seems to complicate everything quite a bit. Therefore we feel that the best solution is to exclude a Delete option. and any node that finds it should get off the list. perform the usual mid-list deletion.

2.6 BROADCAST UPDATE POTENTIAL PROBLEMS

2.6.1 Virtual Memory Tables

A potential problem by caching virtual memory tables. is pending writes. I.e. writes that have used the virtual memory tables for a successful physical address generation. but they have not yet updated the memory image. A node must not allow the tables used for a pending write to be invalidated before the write has updated the memory image. Below we present three methods that are able to solve the problem.

2.6.1.1 Retry Memory Mapping

If (TLB_Flush and Pending_Write) Restart_Instr(Pending_Write)

If a node receives a request that in some ways demands a partial or complete flush of a local processor's Translation Lookaside Buffer. it restarts the pending write instruction. Now the physical address mapping is checked again. If no change has been done to the mapping. the write continue to update the memory image. If there has been a change. the pending write must be inhibited and the processor must rebuild the mapping.

2.6.1.2 Reject TLB_Flush Request

If (TLB_Flush and Pending_Write) Reject (TLB_Flush)

Restarting the pending write may be painful in some cases. Another method is to Reject the incoming TLB_Flush request. I.e. send the request back to the master and let the master issue a new TLB_Flush request later. I the meantime some measures can be taken by the target node to assure that when the TLB_Flush arrives next time. it will not be rejected again.

2.6.1.3   Delay TLB_Flush Execution

If (TLB_Flush and Pending_Write) Delay (TLB_Clear) Until (NOT Pending_Write)

In some cases it might be advantageous just to delay the execution of the TLB_Flush until all pending writes are finished. This is a potential deadlock situation. and must be watched very carefully.

A potential deadlock may occur if the master of the TLB_Flush is the slave of any of the pending writes. Most of these can be solved by implementing control that lets requests be processed independently of any non-completed operation. Another serious situation arises when the master of the TLB_Flush is the slave of another TLB_Flush. Most (all??) of these can be solved by letting requests be processed out of order.

2.6.2 Logical Level Deadlocks

We are not able to see that the broadcasts we propose. introduce any new deadlock situations The reason seems to be that the broadcasts are done in a structured way using the memory controller as a syncronizing barrier. This is the only new thing in this proposal. List insertion and deletion has no changes done to them.

3 REJECT RESPONSE

3.1   PURPOSE

The purpose of the Reject response is to return a SCI operation. that needs some kind of complexity from the slave to execute properly. to the master to simplify handling.

3.2   BACKGROUND

A slave must to a certain degree have an overview over the requests and the responses that it has acknowledged but not yet processed. Depending on how the queues are implemented and how the node acknowledges transactions. It must detect situations that does not conform to what is allowed on SCI and solve them.

For instance we can assume that the SCI definition specifies that a node must have a queue for requests and another for responses. To get better queue efficiency in a space critical design. this might be implemented as the same physical queue. If the node receives a request at the same time as it is waiting for a response. it enters a potential deadlock situation.

The node can solve this by itself by allowing the queue to be processed out of order. This implies quite a bit complexity to determine what is allowed to be processed out of order. Or the request can be returned (Rejected) to the master with the implicit message of trying again. This seems much simpler. Two CPUs interrupting each other at the same time. might encounter a situation like this.

Another situation arises when one master (for example low priority) floods a slave with transactions so that no other nodes (higher priority?) gets access. If the SCI definition specifies fair servicing. this is not allowed. There must be some kind of protocol here to avoid this.

A simple solution seems to be for the slave node to know the origin of all the requests in the queue, and to monitor the incoming transactions. If it detects that one low priority node is flooding itself and that it has to busy a high priority transaction, it grabs one of the flooding transactions and return (Reject) it to the master. The implicit message is to keep quiet for a little bit. A low priority DMA device and a high priority CPU may encounter a situation like this at the memory.

3.3 PROPOSAL

We propose a Reject response which is a response where the corresponding request is not processed.

4 SEQUENCING

4.1 PURPOSE

The purpose of conveying different ways to provide sequence control, is to be able to provide both simple (low performance) and complex (high performance) sequence control, but still using the same simple and high performance scheme for SCI operations not needing sequence control.

4.1.1 Background

SCI itself does not guarantee that the sequence of transactions received by a slave is the same sequence as transmitted by the appropriate master. However, there are some operations where the sequence of which they are carried out at the client is important.

Let us assume two uncached accesses to the same memory location. Before we start, the memory location contain the value M. The first access to be executed is a write with the value W. The second access to be executed is a read. If the read is executed before the write, an incorrect M will be returned instead of W which is correct.

The sequencing is also important in some SCI-list operations. In Broadcast Update it is necessary that a 'later' update does not by-pass an 'earlier' update any time during the list traversal.

4.1.2 Proposal

We propose that the master is responsible for the sequence control, and that we use the request/acknowledge transaction as the sequence control mechanism. To be able to handle the Reject response, we propose a Reject Allowed bit in the request and a Processing bit in the acknowledge to optimize the sequence control. Without the Reject response, these bits are not needed.

4.1.3 Behaviour

Each master has a maximum number of possible outstanding requests, equal to the number of different identities. The master is responsible to do the checks to make the decision whether or not sequencing is needed. In case it is; the master takes the action that guarantees that the 'early' operation is executed before the 'late' operation.

The request/acknowledge handshake is supposed to be the sequence control mechanism. But with the possible Reject response involved, this is not enough.

So with each request we have a bit, Reject Allowed, which is a indication whether or not this is a sequence sensitive transaction. The slave acknowledges with either Processing or Not Processing. It can either static acknowledge Processing or Not Processing, or it may set this status according to the request (if it has time).

The intention is that if a slave answers with Processing, it is not allowed to perform a Reject response. All potential deadlocks and servicing and so on must be solved by the slave itself. However, if it answers with Not Processing, it is free to issue a Reject response on that transaction at all times (regardless of the Reject Allowed bit).

≡*SCI* —— Packet formats ——

Read
Operation

Write
Operation

Cache
Coherence
Operation

Request | Header
Address
CRC

Ack | Header
CRC

Response | Header
Address
Data
CRC

Ack | Header
CRC

Request | Header
Address
Data
CRC

Ack | Header
CRC

Response | Header
Address
CRC

Ack | Header
CRC

Request | Header
Address
CRC

Ack | Header
CRC

Response | Header
Addres
CRC

Ack | Header
CRC
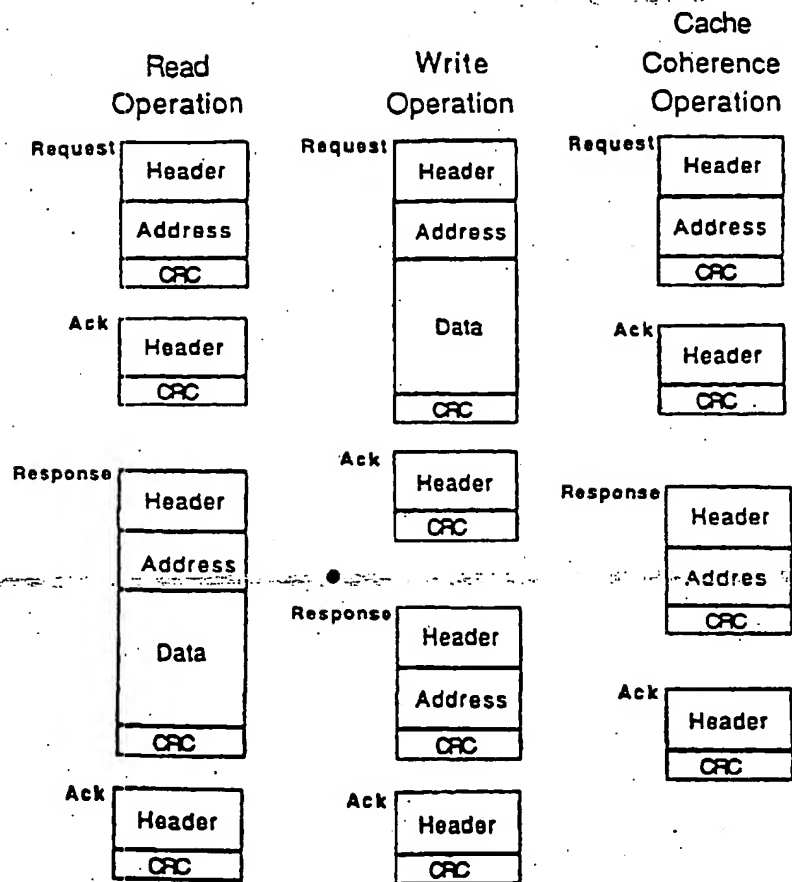
E.D.VA. CA 00-524
1 206793

$\equiv SCI$ —— Header format —— ⚪⚪✳⚪

```
0                                                              15
┌──────────────────────────────────────────────────────────────┐
│                          Target                                │
├──────────────────────────────────────────────────────────────┤
│                          Source                                │
├────┬────┬────┬───┬───┬───┬────┬────┬────┬───┬───┬───┬───┬──┬──┬──┤
│Ack │Bsy │Old │Tp0│Tp1│Go1│Go2 │Go3 │Go4 │Ra │Rp │Su │Pd │  │  │  │
├────┴────┴────┴───┴───┴───┴────┴────┴────┴───┴───┴───┴───┴──┴──┴──┤
│                 Command & Sequence number                      │
└──────────────────────────────────────────────────────────────┘
```
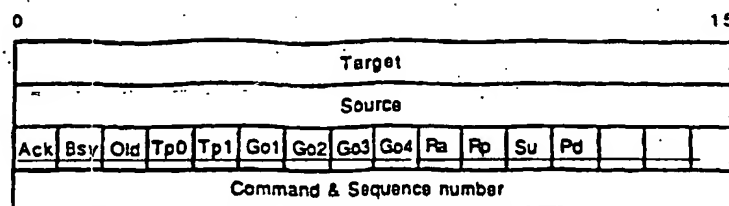
Ack	Set by slave in acknowledge packet
Bsy	Set by busy node
Old	Set by ring cleaner
Tp0-Tp1	Transaction priority (4 levels)
Go1-Go4	Request priority (4 levels)
Ra	Reject from slave's processing queue allowed
Rp	Transaction is Received and Processed
Su	Used during startup and node ID assignment
Pd	Packet Deletion

≡SCI —— Arbitration ——

Node wants to transmit

Node wants to transmit

Go=1 — A — Go=1

Go=0 — F (Node wants to transmit)

B

Go=1

Go=0 — E

C

Go=0 — D — Go=1

Node is transmitting

Go=1 — A — Go=1

Node may start to transmit — F

B — Node wants to transmit

Go=1

E

C

Go=1

D — Go=1

Node is disabled

≡SCI —— Arbitration —— ◯◯⋇◯

Go=0   A   Go=0

Node is transmitting   F   B   Node wants to transmit

Go=1   Go=1

E   C

Go=1   Go=1

D

Node is disabled

A

Node is disabled   F   B   Node may start to transmit

Go=1   Go=1

E   C

Go=1   Go=1

D

Node is disabled

≡ *SCI* — Priority Arbitration

Go4-Go1=0111   A   Go4-Go1=0111

Node wants to transmit
Priority Level 3
Node is enabled Go3=0   F

Node wants to transmit
Priority Level 4
Node is enabled Go4=0   B

Go4-Go1=1111

Go4-Go1=0001

E

C

Go4-Go1=0001   D   Go4-Go1=1111

Node is transmitting
Priority Level 2

Go4-Go1=0111   A   Go4-Go1=0111

Node is disabled
Go3=1   F

Node is enabled
Go4=0   B

Go4-Go1=0111

Go4-Go1=1111

E

C

Go4-Go1=0111   D   Go4-Go1=1111

Stop all nodes with priority < 4
Send priority increase packet

≡SCI — Priority Arbitration

Go4-Go1=0011   A   Go4-Go1=0011

Node is enabled
Go2=0   F

B   Node is enabled
and transmitting
only one packet

Go4-Go1=0000   Go4-Go1=0000

E

C

Go4-Go1=0000   D   Go4-Go1=0000

Node has nothing
to send

Go4-Go1=0011   A   Go4-Go1=0011

Node can start
transmitting   F

B   Node has
nothing to
send

E

C

D

Node has nothing
to send

$\equiv SCI$ —— Request-Ack

≡ *SCI* ── Response - Ack ──
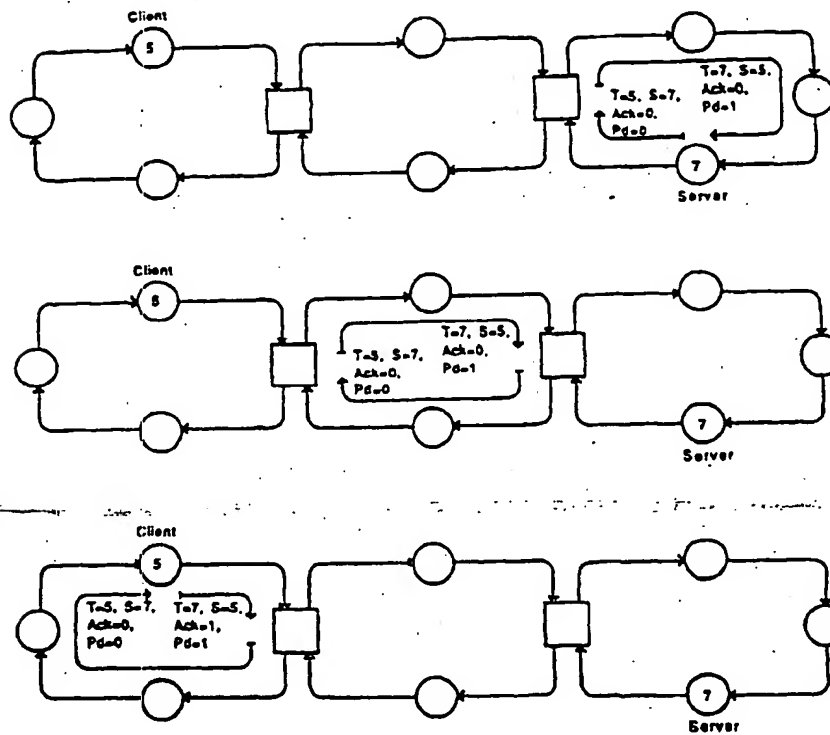
$\equiv$*SCI* —— Response - Ack ——

$\equiv SCI$ —— Node Interface —— ◯◯✕◯

CPU/Cache

SCI Registers

Fault Retry Screen

Packet Manager

| req | response | req | response |

May
reject
from
this
queue

Slave
Fifo

Master
Fifo

Bypass Fifo

E.D.VA. CA 00-524
! 206803

≡ *SCI* ———————— Reject ————————— ◯◯※◯

Req : Reject not allowed
Ack : May be rejected
Wait for response

Slave cannot
guarantee
processing

A: $Ra=0$, $Rp=0$
B: $Ra=0$, $Rp=0$
C: $Ra=0$, $Rp=0$
D: $Ra=0$, $Rp=0$
E: $Ra=0$, $Rp=0$
F: $Ra=0$, $Rp=0$

Req : Reject not allowed
Ack : Will not be rejected
May issue new request

Slave guarantees
processing

A: $Ra=0$, $Rp=0$
B: $Ra=0$, $Rp=0$
C: $Ra=0$, $Rp=1$
D: $Ra=0$, $Rp=1$
E: $Ra=0$, $Rp=1$
F: $Ra=0$, $Rp=1$
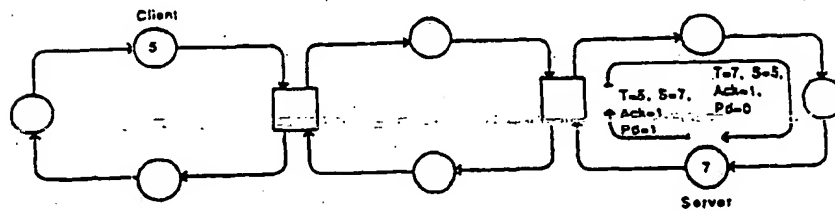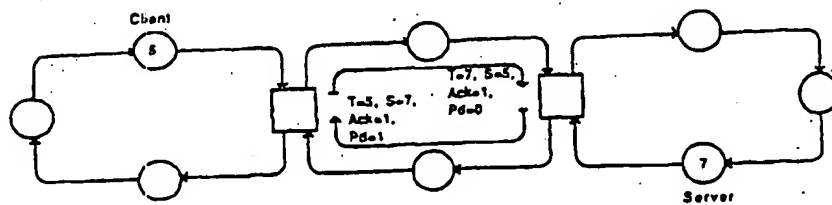
E.D.VA. CA 00-524
I 206804

$\equiv SCI$ —— Broadcast Update ——

Broadcast Update Start

C → C — C → C — C → C

C → C

Broadcast Update Request

Broadcast Update Completion

Broadcast Update Start

D → C — C → C — C → C

D → C

Broadcast Update Request

Broadcast Update Completion

Broadcast Update Completion

P → P — I → I

D → D

Broadcast Update Start

Broadcast Update Request
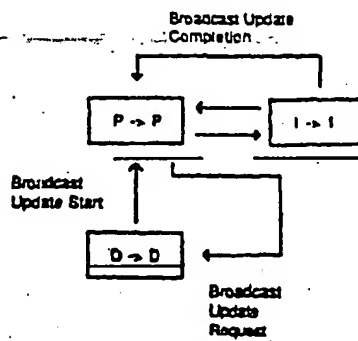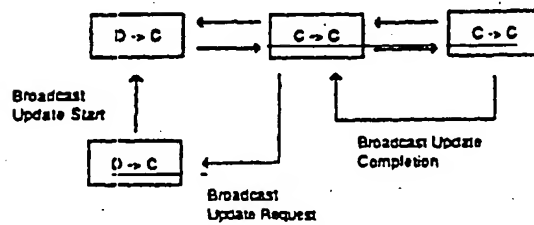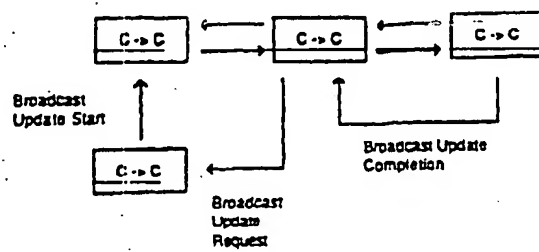
E.D.VA. CA 00-524
I 206805

≡ *SCI* — Broadcast Invalidate — ⬭

New state : Nothing Valid (N)

| C → P | | C → I | | C → N | | C → N |

Broadcast
Invalidate Start

| C → D |

| D → P | | C → I | | C → N | | C → N |

Broadcast
Invalidate Start

| D → D |

After Broadcast Update (item size = cache line size) :

| P → C | | I → C | | N → C | | N → C |

| D → C |

E.D.VA. CA 00-524
I 206806

Scalable Coherent Interface

Ernst H. Kristiansen
Knut Alnæs
Bjørn O. Bakka
Marit Jenssen

*Dolphin Server Technology A.S.
Oslo, Norway

Abstract

The Scalable Coherent Interface (IEEE P1596) will establish an interface standard for very high performance multiprocessors, supporting a coherent-memory model scalable to systems with up to 64K nodes. This Scalable Coherent Interface (SCI) will supply a peak bandwidth per node of 1 GigaByte/second. The standard will facilitate assembly of processor, memory, I/O and bus adapter cards from multiple vendors into massively parallel systems with throughput high above what is possible today.

The SCI standard will encompass two levels of interface, a physical level and a logical level. The physical level will specify electrical, mechanical and thermal characteristics of connectors and cards that meet the standard. The logical level will describe the address space, data transfer protocols, cache coherency mechanisms, synchronization primitives and error recovery. In this paper we will address logical level issues such as packet formats, packet transmission, transaction handshake, flow control, and cache coherence.

1 INTRODUCTION

The Scalable Coherent Interface (SCI) Project started in November 1987 as a study group under the Microprocessor Standards Committee (MSC) of the Technical Committee on Mini and Microcomputers in the IEEE Computer Society. Paul Sweazey was the chairman for the study group that used the working name SuperBus. In July 1988 the status of the study group changed to project and the name Scalable Coherent Interface was adopted. Chairman for the project is David B. Gustavson [1,4,5].

The objective of the SCI working group is to define an interconnect system which *scales* well as the number of attached processors increases, provides a *coherent* memory system, and defines a simple *interface* between modules.

In order to achieve our goals, we quickly discovered that a traditional backplane bus could not be supported by this standard. Today's buses are limited by the distance a signal must travel and the propagation delay across a backplane. In asynchronous buses, the limit is the time needed for a handshake signal to propagate from the sender to the receiver and for a response to return to the sender. In synchronous buses, it is the time difference between clock and data signals which originate in different places.

Transmission lines in backplanes are disturbed by connectors, and variations in loading as the number of inserted modules varies, makes the use of a backplane bus less attractive. In addition, a backplane bus can only service one request at a time and therefore becomes a bottle-neck in multiprocessor systems.

The SCI working group attempts to solve these problems by defining a radically different interconnect system. We are defining an interface standard which enables a system integrator to connect his boards into a network of many different configurations. These configurations may range from simple rings to complex multistage switching networks.

The interface standard defines a point to point communication between neighbour nodes reducing the non-ideal transmission line problems. A point to point link will consist of differential ECL lines, allowing high speed transfers of 1 Gbyte/second. A link is 2 bytes wide making the interface very simple. Small packets carry data from node to node across these links. Buffering in the node interfaces allows many simultaneous requests, making SCI well suited for high performance multiprocessor systems. The SCI standard allows up to 64K nodes to be connected to a network and, should provide the next generations of computers with sufficient interconnection bandwidth.

Cache coherence is an important part of the proposed standard. Current mechanisms prove insufficient when the number of processors increases dramatically. This calls for a new

* Address: Dolphin Server Technology A.S.
P.O. Box 52 Bogerud

approach to the cache consistency problem. The SCI working group is defining a distributed directory scheme where processors sharing cache lines are linked together by pointers. The benefit is a cache coherence mechanism which scales well.

High volume products, using the SCI standard, is expected to become available in the second half of the 1990's. Figure 1 gives a rough estimate for volumes of board level products [2] in the future.
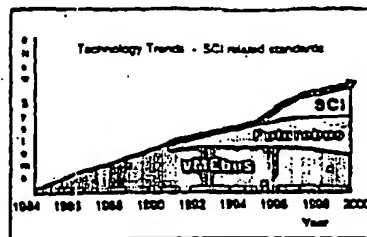
Figure 1. Technology trends

The following sections will provide more insight into the solutions which the SCI working group is currently pursuing. The next section will describe different configurations for an SCI system and emphasize on interfacing to different networks. The packet format and packet transmission is described in section three. In section four we will focus on the mechanisms for packet flow control. Section five gives a brief overview of the cache coherence model. Finally, we will briefly discuss the standardized Control Status Registers space and the status for realization in silicon.

2 CONFIGURATIONS

SCI supports multiple configurations, ranging from simple, low cost implementations to high performance, high cost systems. An important property of SCI is that it includes hooks to allow several different implementations to reside simultaneously in a system. This is done by separating the interfacing node from a transporting network. A view of a system is illustrated in Figure 2.

2.1 SCI viewed by a node

An SCI node receives a steady stream of data and transmits another stream of data. These streams

consist of SCI packets and idle symbols. A node is responsible for operating on these packets and idle symbols according to the SCI standard. To do that, a node may have the construction shown in Figure 3.
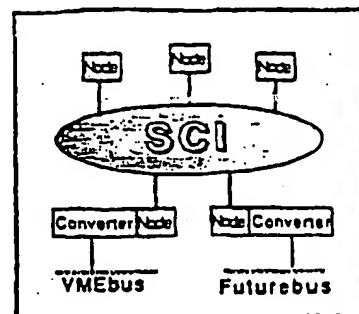
Figure 2. SCI Configuration.

When there is no traffic on the SCI network, a node receives idle symbols. Since the utilization is zero in this case, all nodes are free to transmit. The idle symbols convey this information to the nodes. In case the node has nothing to send, the output consists of idle symbols only.
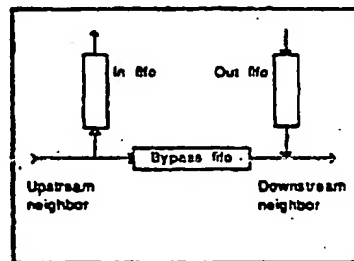
Figure 3. SCI interface.

When a node receives a packet, it checks the packet destination. If the packet is not destined for that node, it is routed to the bypass fifo and transmitted again. However, by this retransmission the node is able to inform the SCI system whether or not it waits for a permission to send. This information is divided between the packet header and the (minimum one) idles separating each

packet. The arbitration, priority and forward progress schemes are enforced this way.

When a node receives a packet which is destined for it (and it is ready to accept it), the packet's main body is routed to the input fifo. It stays there until the node has time to process it further. The packet's header is routed to the bypass fifo and assembled into an 'echo' packet. The echo is transmitted and picked up by the packet's sender. This method is used to assure that the arbitration, priority and forward progress schemes are independent of the physical position of any node.

The SCI system uses idles, packet headers, and echoes to permit transmission of packets. A node which is granted network access and which has an empty bypass fifo, is allowed to transmit a packet. Since many nodes may have network access simultaneously, multiple nodes may transmit at the same time. This contention is solved either by buffering in the network or by filling the bypass fifo of the transmitting node(s).

2.2 SCI as a network

SCI can be configured in many ways. However, there are two basic structures - the ring and the switch. The ring implementation is the simplest. In a ring, nodes pass packets to their neighbour. In such a structure there are no active components except the nodes. This means that the nodes themselves have to control the arbitration, priority and forward progress schemes.
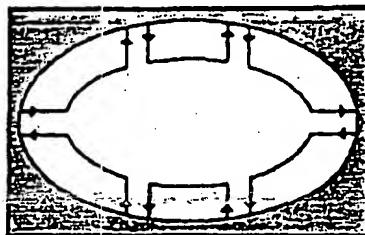
Figure 4. Ring network.

A switch looks at the destination address and route the packet to the destination at once. A switching structure can be of various complexity, including full crossbar switches and butterfly switches. In a switching structure priority and forward progress schemes can be done by active switches. The node interfaces are the same in both a ring and a switch implementation.
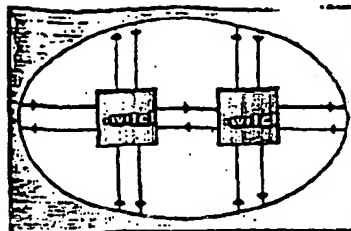
Figure 5. Switch network.

2.3 Interconnect to other buses.

Another important feature of SCI is the ability to interface to other buses. To a certain extent, SCI operations and cache states are defined to accommodate other buses.

A bus convener is defined with a unique address. The bus convener node is responsible for converting SCI operations into native bus operations. Two cases are handled with special care, bus locking and cache coherence.

Most backplane buses accommodate a unique read-modify-write operation to manipulate semaphores and other critical data. During the read operation, a lock signal is asserted inhibiting the use of the bus until the data is written. Since SCI is defined with a four phase transaction protocol with no guaranteed delivery in order, the lock is executed as a single SCI operation.

Some bus protocols also incorporate a cache coherence scheme. Most of them use a snooping scheme where bus interfaces monitor all bus activity and update their cache states accordingly. In SCI this is not possible since it has a different interconnect structure.

2.4 Scalability

A significant aspect about SCI is scalability. It will be possible to have a simple, cheap system with the same basic properties as a high performance one. To achieve this, a large and important task of the SCI working group is to assure that enough, but not too much, functionality is included in the standard.

A simple and cheap system can be connected as a ring. The traffic will only consist of single priority level packets. This results in round robin arbitration. A requesting node has only one packet outstanding at any time, but it supports separate request and response queues. A responding

node is only able to handle a single request at a time. If the node is busy, requests for that node is kept waiting by circulating in the ring.

A more complex, but still fairly cheap, system can be a switching network built of elements like the butterfly switch. The network is hardwired and a node can only be plugged into a certain location. This kind of network is able to handle more traffic and multiple outstanding requests are supported by a requesting node. There is no round robin arbitration scheme but multiple priority levels instead. The switches uses these priorities and the traffic load to decide the packet routing. The switch issues idle symbols according to packet priorities and network loading to control the packet flow. In case of overflow, requests are returned to the requesting node to slow down the packet sending rate.

The most complex system considered is a combination of rings and self configuring switches. The rings are used between nodes which require low latency and where the ring bandwidth is sufficient. The switches are used as interconnects between local rings. The switches are intelligent and support a dynamic network where a node can be plugged into any socket. The network also supports live insertion and withdrawal.
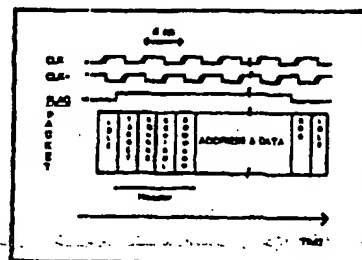
Figure 6. Packet format.

3 PACKET FORMAT

Figure 6 shows the current packet format. The width of a packet word is 16 bits. In addition, a flag indicates that a packet is being received or transmitted. Each word in the packet is clocked with a differential clock line. A node receives 2 bytes at a rate of 500MHz resulting in a network bandwidth of 1 Gbyte/second.

A packet consists of three main sections: a header section, an address and data section, and an error check word. The first 16 bit word of the header

contains the id code of the final receiving node. By looking at the first word of a packet, a node can quickly determine if the packet is addressed to that node. During routing through an SCI network, intermediate nodes and switches look at the target word to determine where to route the packet. The second word of the packet contains the id code of the sender enabling the receiver to return a response back to the correct sender.

A detailed header format is shown in Figure 7. The control word of the header controls packet flow and network access. Priority arbitration is supported with round robin arbitration on each level. Flow control and arbitration will be discussed in more detail in a later section. The command word of the header contains the transaction command and a sequence number. The sequence number is a tag to identify a packet. A node connected to an SCI network may send many requests (currently 256), before a response is received. This transaction pipeline can cause responses to be returned out of order, and therefore a sequence number is needed to identify a response with the corresponding request.
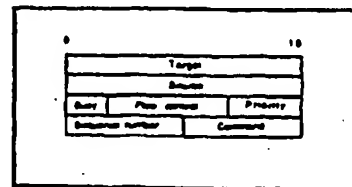
Figure 7. Header format.

The command field contains the command a receiver must execute. In a typical SCI network, a command is applied to a cache line. Suggested cache line size is currently 64 bytes. However, manipulations on smaller and larger data sizes are also supported. The commands can be divided into cache coherence operations, lock operations, DMA operations, and I/O registers operations. The cache coherence operations manipulate a linked list structure used to maintain a coherent memory image.

The target word and the three first address words define the 64 bit SCI address. The data part may contain data ranging from 16 to 256 bytes. When a packet is transmitted, a cyclic redundancy code (CRC) for the packet is computed, and this code is attached after the last word of the packet. The CRC that will be used is a "serial-parallel" version of the 16 bit CCITT-CRC.

3.1. Packet reception

In an SCI network, a node is addressed by a 16 bit identification code. This allows 64K nodes to be attached to the network. As explained above, the 16 bit target id code is located in the first word of the packet header. This allows for easy detection, and a decision to pick up the packet can be made fast. When a node's input flag is asserted, it knows that a packet is entering the node interface. If the target id of the packet matches the id code of the node, and the node's input fifo is empty (see Figure 3), the packet will be accepted into the input fifo. While the packet is being accepted, a CRC for the packet is computed. When the packet has been received, the computed CRC code is compared with the CRC code at the end of the packet. If they match, the reception is completed. Also, while the packet is being accepted, the contents of the bypass fifo may be transmitted.

If the input fifo is not empty, the busy bit in the control word of the incoming packet is set, and the target and source words are swapped. The busied packet is then sent back to the original sender. The target and source fields are swapped such that the original sender can easily detect that the transaction was busied.

3.2 Packet transmission

A node may transmit if the bypass fifo is empty (see Figure 3) and the node is granted network access through the flow control mechanism. Before transmission, the packet is put into the output fifo.

Transmission starts by putting the target word onto the output word and setting the output flag high. The output flag is high as long as the packet is being transmitted. While the node is transmitting, a CRC code for the packet is being computed. This code is attached at the end of the packet after the output flag goes low. If a packet is entering the node interface during transmission, and the packet is not for this node, the packet is put into the bypass fifo until the transmission is done. The size of the bypass fifo must therefore be the same as the maximum packet size to avoid fifo overflow.

3.3 Transaction handshake

SCI supports a transaction pipeline up to 256 transactions deep. This means that a node may send up to 256 requests without waiting for a response. The current transaction handshake consists of request, response, discard and complete transactions as shown in Figure 8. When the request is transmitted, it is tagged with a sequence number. The id code of the sender and the sequence number uniquely identify a packet in the SCI network. When a receiver accepts a

packet, the sequence number in the request packet is saved. The receiver will add this sequence number to the response packet when the response is transmitted back to the sender.
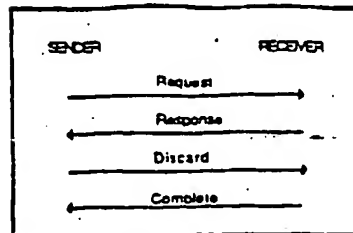
In a packet switching network like SCI, transmission errors cause many problems. Usually, transmission errors are handled by a time-out mechanism allowing the sender to retry a transaction if no response has been received within a time-out interval. However, retried transactions can cause severe problems when pipelined transactions are used. When pipelined transactions are retried, the transactions are no longer issued in order and this may cause incorrect operation. Even if the retried requests do not cause incorrect operation due to out of order execution, duplicate retried requests can. If a transaction is accidently processed twice, incorrect operation may be the result.

Usually, the out of order execution problem is solved by allowing the receiver to accept packets in sequence number order only. This mechanism is known as sliding windows and works well when the number of nodes is small or the complexity can be large. However, SCI supports up to 64K nodes each with a 256 deep transaction pipeline, and it would be too complex to implement a sliding window protocol. Instead, we rely on the sender to make sure that transactions are executed in order. This means that the sender must detect when retried request can cause sequencing problems. In this case, the sender does not issue a new request before a response on the previous request has been received. This case can be optimized if we use an optional acknowledge on the request. This allows the sender to issue a new request after the acknowledge instead of waiting for the response.

Duplicate retried requests may also cause problems and must be processed correctly. When a receiver accepts a request, it adds a transaction identifier to a retry screen list as shown in Figure 9. If the response on that request is damaged, the original sender gets a time-out and retries the request. However, the retried request is found in

the retry screen list and is therefore ignored. When a sender receives a response, it issues the discard transaction which removes the transaction identifier from the retry screen list. This transaction is confirmed by the complete transaction. This handshake and the use of a retry screen guarantees that the same requests will not be executed twice due to retries.
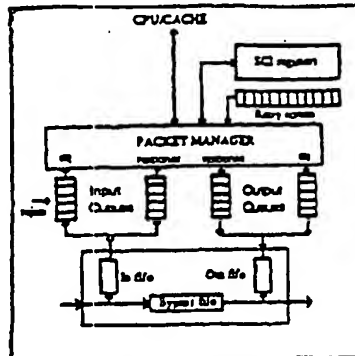
Figure 9. Node Interface.

4 FLOW CONTROL

In SCI, flow control of packets is needed to maintain high throughput and to resolve potential problems as many packets exist in the network at any time. The flow control issues discussed in this section are arbitration, deadlocks, servicing, and congestion.

As explained earlier, a node may transmit when its bypass-fifo is empty. This means that up to 64K nodes may start to transmit at once allowing 64K packets to exist in the network. However, it is possible that a node connected to a ring may not be able to transmit because its bypass fifo will never be empty. In order to avoid this an arbitration algorithm is developed which ensures that all nodes have access to the ring. Our current algorithm is based on a priority scheme with round robin arbitration on each priority level. Up to 75% of the network bandwidth may be used for high priority transactions while the rest can be used for fair accesses. The priority level of a transaction is coded into the control word of the packet header as shown in Figure 7.

Any other node which wants to transmit and which has a higher priority, marks the header of a passing packet. This informs the receiver of that packet that

another node with a higher priority wants to transmit. The receiver then issues idle symbols which stop other nodes on lower levels from transmitting and start nodes on the new priority level. The idle symbols are cycles between packets when the flag is low. By using idle symbols and information in the packet header, the above arbitration scheme can be implemented.

Another objective of flow control is to prevent deadlocks. An incoming request may prevent a response from being serviced, thus creating a deadlock situation. In order to solve potential deadlocks, separate request and response queues are added to each input and output fifo as shown in Figure 9.

Servicing and congestion are based on the objective that no transaction should ever be prevented from getting through to a destination. This is especially important in a multistage switch network where many nodes may compete for the same resource. Currently a reject mechanism allows a node to throw out transactions from the input queues in favour of other transactions which have problems getting through the network. Also, a node may selectively pick up the transactions it wants to process in order to guarantee that a node is serviced.

5 CACHE COHERENCE

High performance processors need local caches to speed up memory access. In a multiprocessor environment, this leads to potential conflicts, because many processors may simultaneously want to cache local copies of shared data.

Cache coherence protocols define mechanisms that guarantee consistent data even if data is cached and modified in local processors. The SCI definition supports a cache coherence protocol which is hardware-based, thus reducing the programmers software effort to secure consistency, and also reducing operating system complexity.

Many existing cache coherence protocols use a snooping technique and rely on operations like broadcast and eavesdropping to guarantee data consistency. In a large high speed distributed system, the broadcast-operation is ineffective at best, and eavesdropping is impossible to implement because it requires a bus common to all processors in the system. Since a highly scalable interconnect system is one of the main objectives in defining the SCI, these and similar mechanisms are inefficient.

Work has been carried out to identify a directory-based cache coherence protocol [6] with distributed properties, where all the nodes with cached copies participate in the control. The

principle is that every sharable block in memory is associated with a list of processors sharing that block. A memory block is usually the size of a cache line which is currently 64 bytes. Every block has a tag which includes a pointer to the processor at the head of the list. Each processor cache tag has a pointer to the next node sharing that cache line. In effect, all nodes with cached copies of a memory block, are linked together by these pointers. The nodes have a forward pointer and a backward pointer to connect them with the previous and next node in the list. The resulting double linked list is shown in Figure 10.
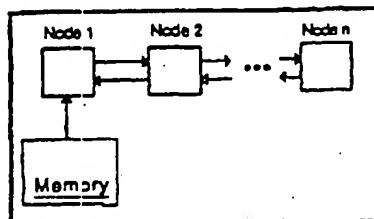
Figure 10. SCI sharing list.

This distributed list concept ensures good scaling properties. Even if the number of nodes in a list grows dramatically, a corresponding increase in memory tag size is not needed. All that is required is two pointer locations associated with every cached block in a node.

The list pointers are actually the network addresses for the processors. When a node accesses memory to get a copy of shared data, it provides memory with its own address. If there is currently no nodes with cached copies, the requesting node is made the head of a new list and memory stores this node-address in the tag for this block. If however, there exists nodes with cached copies of data, the pointer to the head of the sharing list is returned from memory to the requesting node, and this node now inserts itself at the head of the list. If the data was locally modified by the previous list head, the new head must get data from the old head, rather than from memory.

The nodes in a linked list will typically have read access to shared data. Whenever a node wants write access, it deletes itself from the list if it were already a part of it, inserts itself at the head of the list, and then purge the rest of the list. Write access is restricted to the head node only.

All bus operations concerning cache coherence are implemented in the standard packet format described above.

6 CONTROL STATUS REGISTERS

The Control Status Registers (CSR) is an important part of the proposed standard. The CSR definitions are essential for all initialization and exception handling. Parts of the CSR must be SCI specific, but the majority of the necessary definitions can be common with other standards [3]. IEEE MSC has approved a request for a standard project for CSR. The CSR standard will be coordinated for Futurebus, Rugged Bus , Serial Bus and SCI. One will also try to coordinate with the ongoing CSR activity for VME-bus.

7 REALIZATION

Realization in real systems is important for acceptance of a defined standard. Therefore the first implementation will be done in parallel with the standardization work.

So far we have done measurements that ensure us that it will be possible to make implementations for 1 Gigabyte/second transfer rate. The length of a maximum datapacket will in the first implementation be limited to 64 byte (i.e. a cache line). The node circuitry will be made as an ASIC in advanced ECL. The actual configuration will be a ring structure with high performance CPU's, large main memory and low performance buses for I/O-functions. We expect to have prototypes working next year.

8 CONCLUSION

This paper has presented an overview of the objectives of the SCI working group, and the solutions which are currently being pursued. Scalability of a system is a key aspect as many high performance computer manufacturers are moving towards large multiprocessor systems. In order to utilize those systems efficiently, a cache coherence mechanism must have good scaling properties. Also, for a system to be both cost effective and to support high performance solutions, it is necessary to separate the module interface from the interconnect implementation.

We feel that our current proposals meet these objectives. The SCI project is moving rapidly and has attracted participants from many of the high-performance computer companies. The proposed architecture appears to be technically achievable based on technology available today.

If you would like to participate in this work, or if you would like more detailed information, please contact one of the authors or the chairman for the project:

David B. Gustavson, IEEE P1596 Chairman
Computation Research Group, bin 88

E.D.VA. CA 00-524
I 207748

Stanford Linear Accelerator Center
Stanford, CA 94309, U.S.A
tel: (415) 926-2863
fax: (415)323-3626
Email: DBG@SLACVM.bitnet

9 ACKNOWLEDGEMENTS

Many people have already contributed to SCI's development; though we cannot list them all, we wish to acknowledge a few contributions which seem to us to be particularly significant.

David B. Gustavson of Stanford Linear Accelerator Center is chairman for the IEEE P1596 (SCI) working group and the driving engine for the standard. He has long experience with standard buses, specially Futurebus and Fastbus.

David V. James, originally of Hewlett Packard and recently of Apple Computer. He has brought great insight into the appropriate system architecture for SCI's needs, from register and I/O architecture to distributed cache coherence and forward progress. David is coordinator for the Logical Task Group and has written the majority of the working documents. He is also the contact man for the coming working group for CSR.

Paul Sweazey, originally of National Semiconductor and recently of Apple Computer started the SuperBus study group ,and he was the coordinator for this until the SCI working group was organized. Paul has also brought a thorough understanding of the cache coherence problem, due to his work coordinating the Futurebus Cache Coherence task group.

Paul Borrill of National Semiconductor, Futurebus Chairman, helped pushing the goals to much higher bandwidths and increased parallelism through the use of switches instead of shared buses.

John Moussouris, a cofounder of MIPS Computers, has provided critical insights into the directions we need to take in order to rendezvous with future technology, has helped put us in touch with the appropriate experts, and has helped expose problems and errors in various models.

Phil Ponting of Cern in Geneva has provided effective and vital communications and redistribution to the many European participants.

Hans Wiggers of Hewlett Packards Laboratories has helped us examine various physical layers, and is considering the implications of an optical fiber implementation of SCI.

References

1. D.B. Gustavson, D.V. James, J. Moussouris and P. Sweazey, "The Scalable Coherent Interface Project (SuperBus)", Draft of August 22 1988

2. P.L. Borrill " VMEbus - The Next 5 Years", VMEbus in Research, October 1988.

3. D.V. James, "Scalable I/O Architecture for Buses", COMPCON 1989

4. D.B. Gustavson, "Scalable Coherent Interface", COMPCON 1989

5. E.H. Kristiansen," IEEE SCI (P1596)", VMEbus in Research, October 1988

6. A.Agarwal, R.Simoni, J.Hennessy and M.Horowitz, "An Evaluation of Directory Schemes for Cache Coherence", 15th International Symposium on Computer Architecture, June 1988.

Four-phase LSI logic offers unique advantages in high packing density and low cost. The system designer, however, must consider partitioning details to optimize data interconnections

Four-Phase LSI Logic Offers New Approach to Computer Designer

Lee L. Boysel
Joseph P. Murphy

Four-Phase Systems, Inc.
Cupertino, California

The application of fourth-generation electronics to computer design has forced new approaches, particularly in minimized and shortened data paths. The newest development, MOS/LSI, calls for a new 4φ logic that focuses the designer's attention on areas not previously critical. Unfortunately, very little information describing actual working 4φ LSI hardware designed for a total system application is available.[1,2,3] What information is available is principally military-oriented with cost being only a minor consideration. This discussion considers not only circuit design as such, but also the strategy of partitioning the system architecture to minimize cost. The device described is the main LSI block of a low-cost fourth-generation commercial computer system which is currently in pilot production.

Lee L. Boysel is founder and president of Four-Phase Systems, Inc. He also serves as overall technical coordinator and is largely responsible for having brought this concept to implementation in an operational computer. He received a B.S.E.E. and an M.S.E.E. from the University of Michigan.

Joseph P. Murphy is manager of Semiconductor Operations for Four-Phase Systems, Inc. where he is responsible for translating design of logical operations into actual integrated circuit form, and was a prime mover in the development of the high concept. He has worked at San Jose State College and the University of Connecticut.

WHAT IS 4φ LOGIC?

Four-phase device design refers to an MOS circuit technique in which the zero and one logic levels are generated by charging and discharging small data-holding node capacitors with a sequence of four clock pulses. The node capacitors (including the fanout capacitance for 10 to 20 gates) are typically only a few tenths of a picofarad. In addition, the devices used for charging and discharging these nodes are manufacturable in sizes no smaller than approximately 50 kilohms which results in a typical logic gate delay of 10 to 20 nanoseconds at a few microwatts of CV^2 power. Due to the small device sizes, gate densities are about five times greater than those of the more common integrated circuit type of logic. Combined with an order of magnitude of improvement in the speed-power product, this makes 4φ design an extremely powerful LSI technique. The disadvantage of using this design—the four continuously-operating clock drivers required—is of minor importance in complete systems such as desk calculators, terminals, and small- to medium-sized computers.

The actual operation of 4φ logic is as follows. The clock inputs, which are a sequence of four pulses (Fig. 1) are connected to a typical inverter gate (Fig. 2) As φ3 comes up, Q_1 and Q_3 are turned on and the output data capacitor, C, is charged to the clock voltage. φ3 then returns to ground level, supplying a potential ground path for discharging C. Next, when φ4 comes up and turns on Q_2, the "one" level left on C will be discharged to a "zero" level

141
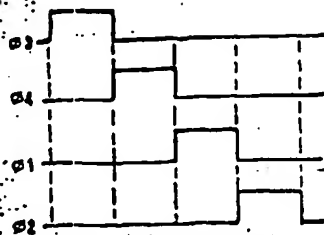
Fig. 1   Four-phase clock inputs
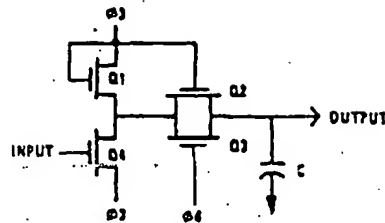
Fig 2   Typical MOS/LSI inverter gates
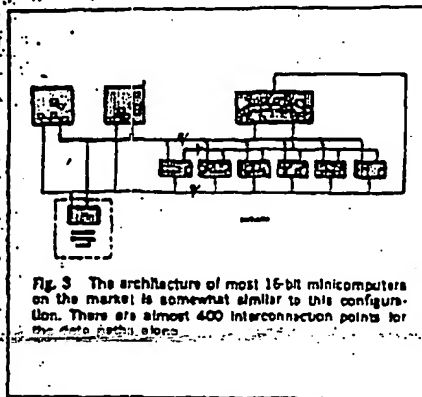
Fig. 3   The architecture of most 16-bit minicomputers on the market is somewhat similar to this configuration. There are almost 400 interconnection points for the data paths alone

Fig. 4   Reorganizing the elements in Fig. 3 and adopting a single bidirectional data bus technique, the number of data lines passing through the critical point is reduced to 16. The data interconnection to the right of this point may now be grouped for integration into an LSI chip

If the input to Q_4 is a "one" level. The output data capacitor is then stable for use in later gate times $\phi1$ and $\phi2$. Note that the conditional discharge gate, Q_4 shown here as a simple inverter, may be replaced by a complex group of AND, OR, and Invert gates.

TYPICAL COMPUTER ARCHITECTURE

A typical 16-bit minicomputer is shown in Fig. 3. The number of registers or exact data flow configuration may be slightly different but most computers conform to this general structure. In examining the structure to determine the type of subsection partitioning which would be best for this LSI machine, it becomes obvious that there are an excessive

number of data interconnection points (approximately 400). Minimization of these data path interconnections is thus the first order of business.

BASIC DATA FLOW INSTRUCTIONS

There are two basic types of instructions to be executed by the computer. First there is data transfer which can transpire between any two points, e.g.

			INTERNAL CONTROLS
ADDXXX	EX. Add S to D	X X 0 X X X	D + S → DO	1 1 0 1 XX
SUBXXX	Subtract S from D	X 0 0 X X X	D - S → DO	1 1 0 1
LDDXXX	Load D on Output	0 1 0 X X X	D → DO	1 0 1 1 0
XORXXX	Exclusive OR S and D	0 1 1 X X X	D ⊕ S → DO	1 0 0 0 1
LDSXXX	Load S on Output	1 0 0 X X X	S → DO	0 1 1 1 0
ORXXX	Logical OR S and D	1 0 1 X X X	D + S → DO	0 1 0 0 1
CMPXXX	Complement S	1 1 0 X X X	S̄ → DO	0 0 1 0 1
ANDXXX	Logical AND S and D	1 1 1 X X X	D · S → DO	1 1 1 1 0
XXXNOS	No Shift	X X X 0 0 0	MSBY LSBY	
XXXSRA	Shift Right Arithmetic	X X X 0 0 1		
XXXSRL	Shift Right Logical	X X X 0 1 0		
XXXSRR	Shift Right Rotate	X X X 0 1 1		
XXXSLQ	Shift Left Quotient	X X X 1 0 0		
XXXSLA	Shift Left Arithmetic	X X X 1 0 1		
XXXSLL	Shift Left Logical	X X X 1 1 0		
XXXSLR	Shift Left Rotate	X X X 1 1 1		

Fig. 5 The eight basic arithmetic and logic operations may be combined with eight shift commands for a combined micro-instruction set of 64. The internal controls are used in Fig. 6

register to register, register to memory, I/O to register, etc. The second involves a logic or arithmetic operation between two data points. The first data point (source data) operates on the second data point (destination data), and the result of the operation is normally stored in the destination register. For example, add Register A to Register B and store in Register B; or subtract memory from Register A and store in Register A.

If the general register and the A/L (arithmetic logic) blocks are grouped together, only 32 data lines would normally be required to interface with the memory, I/O, and random control logic blocks. In Fig. 3 note that for both data flow cases, the data flow into or out of the four major areas was mutually exclusive, meaning that only one 16-line bidirectional data bus line is actually required between the register, control, memory, and I/O areas. At first glance (see Fig. 4) it would appear desirable to combine the entire A/L and register areas on one LSI chip having only 16 data line pins. Before pursuing this approach further, however, it is necessary to see whether the number of control logic lines required for this type of partitioning is excessive. Obviously, every attempt must be made to code the control line inputs and minimize pin count.

REQUIRED FUNCTION AND CONTROL LINES

The instructions required of the A/L section must first be selected. The most commonly used logic and arithmetic operations as well as a variety of shift combinations are noted in Fig. 3. An A/L Op Code input of six lines supplied from the random control logic will allow 64 combinations of arithmetic, logic, and shift microinstructions to be executed. In addition, during the execution of these instructions, a set of status signals (commonly known as the condition code or CC) is generated, which indicates the result of the operation; e.g., result of subtraction is zero. The most commonly used status bits are carry, zero, sign, overflow, and least significant bit shift output (used for multiply). These five lines go to a status or CC register in the random control logic.

In addition to the A/L Op Code and status lines, a set of data register address and control lines must be supplied from the random control logic. The six data registers, used as either source or destination registers, may be addressed by decoding a 3-bit source and destination address code. Since eight decoder combinations are available, it is convenient to assign an artificial zero register which can be used for clearing the working registers by loading zero. In

143

desirable for use in increment and decrement instruc-
tions. Addition of a destination Store control, an A/L
Write for gating external data into the A/L area,
and an A/L Read for gating data back out on the
main bus rounds out the list of data path controls
at nine lines.

Adding six lines for power, ground, and a 4φ
clock gives a total pin count of 42 and a complexity
level of approximately 1500 to 1600 MOS gates for a
16-bit word. The resulting 40-to-1 gate-to-pin ratio is
exceptional, but this complexity level would pre-
ently require a 200 x 200-mil LSI chip which is not
now practicable. For this reason and to make this
device more universal, an 8-bit section was selected
and designed to allow any number of these circuits
to be connected to form a parallel 8, 16, 24 or 32-
bit word. Note in Fig. 6 that the most significant byte
(MSBY) and least significant byte (LSBY) permanent
control lines, when active, change the first or last bit
circuitry to correspond to the first or last bit of the
full computer word. An example would be a shift
left arithmetic operation in which the most signifi-

Fig. 7   A 1-bit slice of the 8-bit LSI array. Data moves
through the slice in four sequential phases with only
one gate level active during each phase. In φ3, the
input data is selected and the instruction control lines
are set up. In the next step, φ4, the carry and borrow
signals are generated. Finally, during φ1, the bit in-
puts and the control inputs are active, and the arith-
metical or logical result is sent to the output buffer
which sets up during φ2

Fig. 6   The final 8-bit computer slice, complete with
logic, registers and control may be serially connected
to form any word length. The circuit required 40 pins,
which is the largest standard package available

cant or sign bit of the 16-bit word remains consta
as the number is shifted around it. Thus the sa
chip will function in any byte location.

IMPLEMENTATION

Now that the partitioning arrangement has be
selected, the detailed circuit design must be co
pleted to see if the partitioning is practical from
circuit point of view. While several circuit tec
niques are available, only 4φ logic[a] appears to gi
adequate packing density at high speed and k
power. Using this type of logic the basic machi:
cycle time is divided into fourths, and during ea
quarter cycle one series of logic gates is active. Th
can be understood more easily by considering o
arithmetic block.

If the current convention of 4φ logic is adopte
the data on the external buses and the control lii
signals must be stable during the third (φ3) at

Fig. 8 This schematic corresponds to the block diagram in Fig. 7. Note that the shift left input (SLI) and the shift left output (SLO) as well as the shift right levels (SRI and SRO) effectively shift the results generated by the instruction logic right or left one bit position. The decoded control lines—d, e, f, and g—are specified by Fig. 5
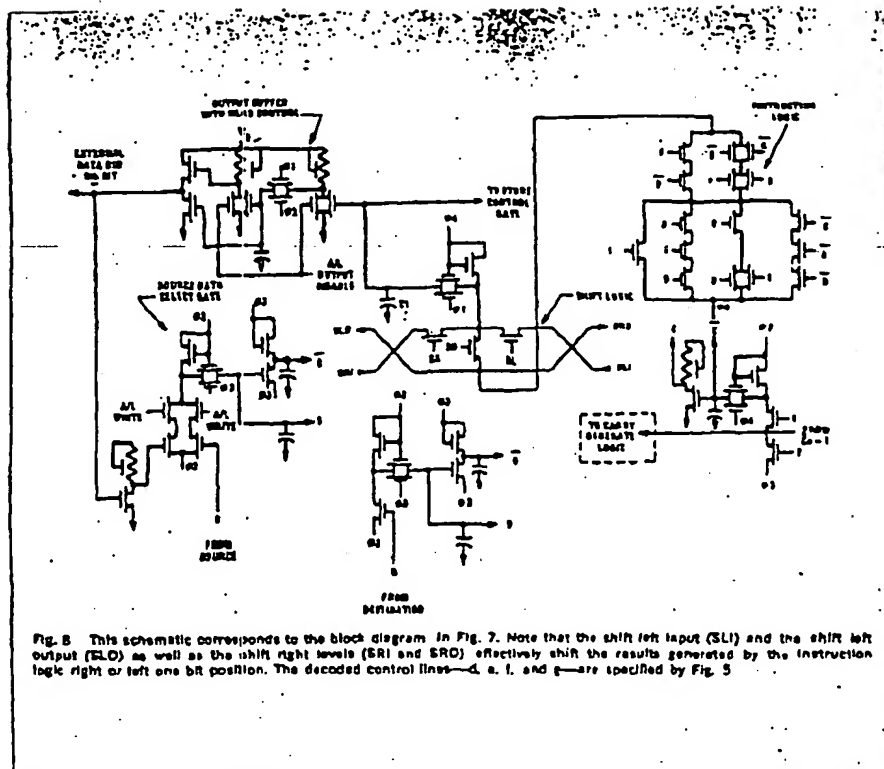
fourth (#4) quarter cycles. This means that the outputs are set up during #1 and #2, normally #2. The basic logic functions, which include add, subtract, AND, OR, shift, etc., must be generated during #1: The add/subtract logic requires that carries be present and stable during this #1 time slot, which forces the carry and borrow signals to be set up during the preceding time slot, i.e. #4. The remaining time slot, #3, is used for data input gating and control logic setup. In Fig. 7, note that the registers at the bottom are loaded at the end of the sequence during #2 and the new data may be read out at the beginning of the next sequence during the #3 cycle. These memory register bits are inactive during #1 and #4, and during this time the register address decoders are set up. Fig. 7 shows a single slice of the circuit and indicates the time slots in which data are being read into the various gate blocks.

Fig. 8 shows the actual circuit schematic (exclusive of the carry and memory circuits). The output buffer

is the type commonly used in MOS designs.[6] Note that the A/L output disable line disconnects the output signal from the bus, allowing bidirectional data to flow into the chip. The main logic block consists of shift gating which allows the data capacitor, C1, to be conditionally discharged through either its own instruction logic block or one on the left or right. This is equivalent to a shift left or shift right command. The instruction logic is a combination of the logic required to perform the eight basic instructions described in Fig. 5. The shift control lines, shift right (SR), shift left (SL), and no shift (NS), as well as the four instruction control lines—d, e, f, and g—are generated from the A/L Op Code. If an add or subtract operation is not selected, g and f interact with the idle carry circuits, minimizing the number of control lines required. Fig. 8 also shows the source selection area, which brings in data either from the external bus or from the internal memory registers.

The carry/borrow circuit Fig. 9 is somewhat unique in that one carry look ahead circuit services

145
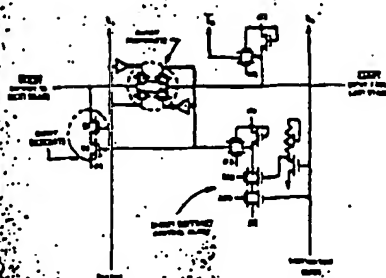
Fig. 9 Carry/borrow circuit shown feeds all eight bits of the LSI array. There are three possible conditions: first, if both S and D are ones, a carry is generated (indicated by a zero output level). If both S and D are zeros, the exclusive OR (carry propagate) gate will not pass the carry from the last stage. If either S or D is a one, a carry from the last stage will be propagated to the next. Direct subtraction is accomplished by merely complementing the 'destination' input.
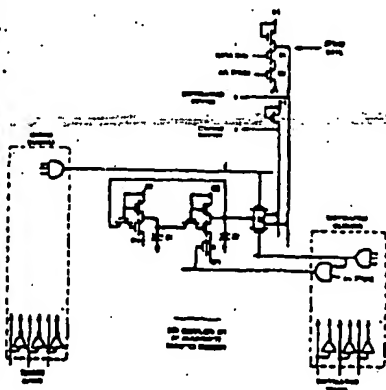
Fig. 10 On-chip register memory cells are basically 2φ shift register bits which have been modified to read/write data in parallel. This cell was chosen for its small size and high speed.

146

destination are one, then a carry is generated in the following stages. If, however, only one bit S or D is on, the carry will only be passed on to the next stage if a carry from the $(n - 1)^{th}$ stage is present. Similarly if both the source and destination bits are zero, there will be no carry to the next stage. If the destination input is complemented, a subtraction occurs with the carry line now becoming a borrow line.

The memory cell, Fig. 10, used for the on-chip register is actually one bit of a 2φ shift register used here for its simplicity and high packing density. In operation, both the source and destination lines for each column, as well as C2, are charged to a logic one level during φ1. During φ2 the data signal on C1 conditionally discharges the C2 capacitor. In addition, if the one-out-of-eight source or destination decoder has selected that register, the corresponding source or destination data line will also be discharged, thus reading out data from that register. Data entry into the register amounts to the same process in reverse. If the A/L Store line is high, the Q4 devices in the register selected by the destination address are turned off or opened, eliminating the normal conditional discharge path for C2. However, a secondary conditional discharge path consisting of Q1, Q2, and Q3 is now in the circuit. Instead of C2 being the complement of C1, C2 will be the complement of the data input. Thus data is written into the register. The source and destination decoders are standard MOS NOR gates.

CONCLUSION

The 4φ logic approach to implementing MOS/LSI technology offers the highest packing densities yet obtained in logic circuits, and at radically lowered cost. These advantages can only be obtained, however, through scrupulously careful attention to the details design of circuitry to obtain low data-connection counts and short data lines. With these tiny chip pin count, device partitioning, and speed-power relationships are also crucial. The design of this arithmetic logic block has illustrated the design sequence required to optimize LSI usage in an actual fourth generation computer system.

REFERENCES

1. L. L. Boysel, "Cutting Systems Memory Costs with MOS," Electronics, Jan. 1969.
2. L. L. Boysel, "Memory on a Chip: A Step Toward LSI," Electronics, Feb. 1967.
3. L. L. Boysel, W. Chan, and J. Faith, "Random-Access Memory Packs More Bits on the Chip," Electronics, Feb. 19
4. D. L. Mann, "The Polynomial Counter Design Technique with Applications to Four-Phase Logic," Computer Design, Nov. 1969.
5. L. L. Boysel, C. Marvin, J. Murphy, and J. Sorenson, "The Way to 'Very' LSI," Electronic Engineer, to be published.
6. L. L. Boysel (with G. P. Certus), "MOS Complex Array Design," Electro Technology, Feb. 1969.
7. L. L. Boysel, "Adder on a Chip: LSI Helps Reduce Cost Small Computing Machines," Electronics, March, 1968.

Random-access MOS memory packs more bits to the chip

Lee Boysel, Wallace Chan and *Jack Faith* of Four-Phase Systems eliminated
the separate feedback for each bit in the design of a 1,024-bit memory;
within three years, the cost per bit could be as low as a fraction of a cent

● To manufacturers of semiconductor memories. the name of the game is putting more and more memory capacity on a single silicon chip. Ground rules usually call for more components for more capacity. But now, what may be the most compact memory array yet produced in quantity—a 1,024-bit random-access memory that fits, with decoding circuitry, on a 150-mil-square chip—achieves its greater capacity without a substantial increase in circuitry. The trick is elimination of separate feedback stages for each bit.

The metal oxide semiconductor memory is based on a modified dynamic memory cell whose stored information must be periodically refreshed. Generally, this is accomplished, as in a dynamic shift register, with a separate charge-refreshing feedback stage for each bit. But in Four-Phase Systems' new design, separate feedback stages are eliminated because a single feedback stage is shared among many bits. The result: a very dense array that occupies 20% less chip area than even a conventional 1,024-bit dynamic shift register, and with four times the random access capacity of monolithic semiconductor arrays now on the market.

The 4,500 active components on the memory chip, which will be available in a low-cost computer system, are organized into 1,024 1-bit words in a 32-by-32 word array. Access time is 1 microsecond (full cycle time is 2 microseconds) and the chip dissipates about 200 milli-

watts. Within three years, the cost per bit will approach a few tenths of a cent, about an order-of-magnitude improvement over the cost of large-scale random access core memories available now. Thus far, large-scale integration of MOS arrays has been too costly for anything but limited scratch-pad applications.[1]

One of the common MOS arrays uses a familiar reset-set, or RS. flip-flop in its memory cell, as shown below. This basic cell has not changed in the five years since it was developed,[2] though new decoding schemes have been developed for moving data.[3,4]

The resistor-like symbol in this and subsequent schematics represents an MOS transistor which functions as a gated impedance. Its impedance is relatively high—100 kilohms. against 20 kilohms in the usual four-phase MOS transistor—and it is switched on and off by its gate connection. The high impedance is obtained by laying out the transistor on the silicon chip with a very long, but quite narrow. channel. The long channel takes up much more space on the chip than those of transistors used simply for logic functions.

In the RS flip-flop, data is stored statically in a cross-coupled pair of NOR gates using transistors with both high and low impedance. With MOS technology, the low impedance transistors occupy a large area because conductance is proportional to the area—and hence the width—of the conducting channel between the
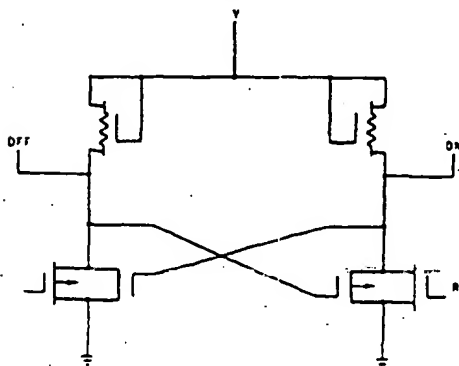
Old standby. Conventional reset-set MOS
flip-flop which stores data statically
takes up too much area and dissipates too
much power, limiting the number of
units that can be put on a single chip.
The resistor-like symbol represents an
MOS transistor which functions as a
gated impedance.

source and drain. This type of cell also dissipa·
lot of power and requires high-voltage and high-cu
line drivers to get data in and out in a typical coinci·
current memory scheme. These factors, combined
the requirement for external decoding, drive, and s
circuits, appear to limit at 256 the number of
on a chip, and the cost from going much below 2
cents per bit.

To break this price barrier. Four-Phase Systems'
design approach is based on the dynamic storage sch
found in a conventional MOS shift-register cell, sh
at the left with the four-phase clocking waveform nee
for operation. In a dynamic cell data, stored in the f
of a charge on a parasitic capacitor. must be periodic
refreshed because it tends to leak off. But the big adv
tage is offered by relatively high transistor impedar
so the area occupied by each transistor is small.

(In this and in the figures to follow, capacitors dr:
with dashed lines are parasitic. Capacitors drawn v
solid lines. although also technically parasitic, have b.
purposely augmented-to increase their value-by enh·
ing the p— region of the silicon in their vicinity.)

The operation of this shift register cell, which actu:
consists of two inverter stages. starts with the f
charging during the first phase of the four-phase cl
cycle of capacitors C_1 and C_2. These capacitors
discharged later in the cycle, but the discharge is o

Dynamic standard. Feeding back one
bit of a dynamic MOS shift-register cell
yields a dynamic memory cell. Data
held on capacitor C_1 (and redundant
data on C_3) is refreshed by charge from
the large output capacitors C_2 and C_4.
The four clock pulses occur during a
single memory cycle. Color tint block
identifies the basic storage unit; gray
tint identifies the feedback circuit that
refreshes the charge on the data-
holding capacitor, C_3.

Reading and writing. A row-address
decoder selects the row in the memory
that's to be read out. This read-row
signal also triggers a delayed write row
circuit that rewrites the readout data,
restoring the voltage level of the signal
on the storage capacitor.

ditional—whether they are discharged depends upon the
data stored in the memory cell.

The precharging occurs during the first clock pulse,
ϕ_3. Transistor Q_1 is turned on and capacitor C_1 is
charged to the input signal voltage level—either a logic
1 or a 0. Simultaneously, Q_2 is turned on and C_2, the
output signal-holding capacitor of the first inverter stage,
is charged to the supply voltage level, V.

Conditional discharging occurs next. When clock pulse
ϕ_3 goes to 0, pulse ϕ_4 comes up, turning on Q_4. If the
charge on C_1 is at the high, logic-1 level, Q_3 turns on
and C_2 discharges to ground through Q_3 and Q_4. But
if the input and the charge on C_1 had been a logic 0,
capacitor C_2 would not have discharged; a charge
equivalent to a logic 1 would have remained.

Passing the signal on C_2 through the other half of
the cell reinverts it, restoring the original signal level
at the cell's output. And tying the output back to the
input, as shown by the solid feedback line in the sche-
matic shown on page 110, converts this one-bit shift reg-
ister into a binary memory element—a dynamic flip-flop
that stores one bit of information.

However, there's a basic information redundancy in
this flip-flop cell—the basic information bit held on C_1
is also held on C_2, although in complement form. C_2
and the second half of the flip-flop bit keep restoring
or refreshing the charge on C_1, which otherwise would

WRITE ROW 0
READ ROW 0
WRITE ROW 1
READ ROW 1
WRITE ROW 2
READ ROW 2
WRITE ROW 3
READ ROW 3

DATA
FEEDBACK
LINE

C_0

C_1

C_2

C_3

ϕ_3

ϕ_4

C_5

$C_3, C_4 \gg C_0, C_1, C_2, C_3, C_4$

DATA
FEEDBACK
PATH

DECODER
SIGNALS

WRITE
ROW

READ
ROW

C_6 C_7

ROW ADDRESS DECODER

THREE-ACTIVE-DEVICE
MEMORY CELL

ROW REWRITE DELAY
(ONE FOR EACH ROW)

E.D. VA. CA 00-524
1 247906

111

863 FH PG 0873
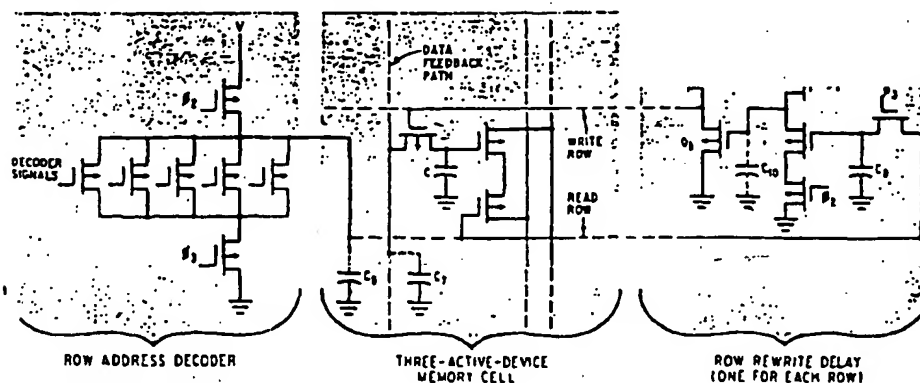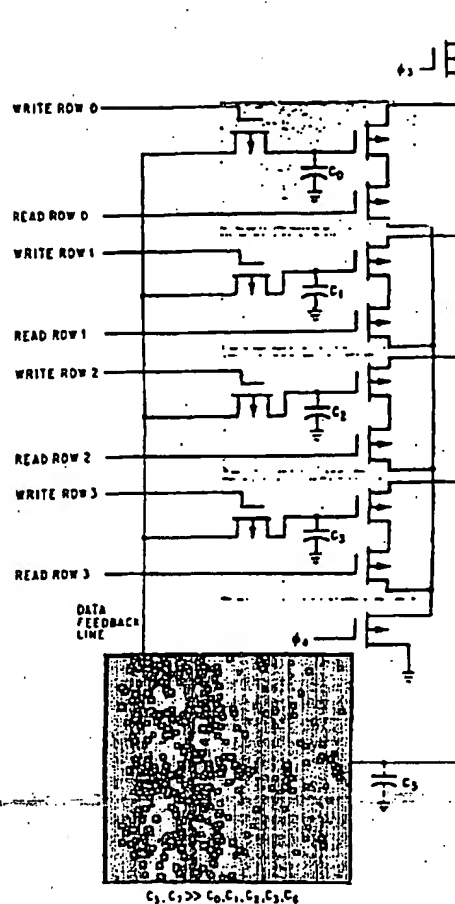
leak off within a few milliseconds. For shift registers and other dynamic circuits on the market today, the maximum charge-holding time is about 100 μsec. This means the minimum clock frequency for restoring data must be 10 kilohertz.

To reduce this information redundancy, Four-Phase Systems shared a common feedback, or charge-restoring, stage among many shift-register bits as shown at the right of page 111. Each memory cell contains three active devices and one data-holding capacitor. In this cell, the second, or redundant, stage of the conventional shift-register flip-flop has been replaced by the single feedback stage shown at the bottom of the schematic. This stage refreshes each of the four data-holding capacitors, C_0, C_1, C_2, and C_3 in sequence under control of the read row gates, which have been added to the basic dynamic cell. Only four memory cells are shown for illustrative purposes. However, in the actual memory, a single feedback stage refreshes a column of 32 cells.

A memory sequence begins when the fourth clock pulse, ϕ_4, comes up. Read row 0 also is high at the same time, so that the signal on C_0 conditionally discharges C_3 previously charged to the supply voltage during ϕ_3.

The charge on C_3 is transferred to C_4 during ϕ_1 and then is inverted and placed on the data feedback-return line capacitor, C_1, during the ϕ_2 pulse. The write row 0 line rises, and during the ϕ_3 pulse the charge on C_1 is transferred to C_0, restoring its original level.

The process repeats itself during the next sequence of four clock pulses. But the read row 1 line is activated during ϕ_4, and C_3 is conditionally discharged by the data-storing capacitor, C_1. This sequence, restoring the last bit interrogated and reading out the next sequential bit, is repeated continually. Thus, four bits of information can be stored with only 18 transistors—three for each cell, five in the feedback stage, and one to charge C_3 at ϕ_3—rather than the 32 required if each bit were stored in a single shift-register cell.

The actual circuit mechanism that generates signals on the read-row-and-write-row lines is shown at the bottom of page 111. Each of the 32 rows is addressed through a standard decoder network consisting, at each row line, of five transistors in parallel. Each transistor is connected to one address bit or its complement.

When the ϕ_2 pulse is present, the output of the parasitic capacitance, C_8, at the output of the decoder network[5] is precharged to the supply voltage. This precharge is retained on the read row line until ϕ_3, when the five-bit address supplied to the decoder discharges C_8 on 31 of the 32 lines through one or more of the five parallel transistors connected between each read row and ground. On the remaining row, the address keeps all five transistors off. Capacitor C_8 on this row retains its charge, and the corresponding read row stays high. During the next clock pulse, ϕ_4, the charge enables the capacitors in each of the memory storage cells to discharge conditionally.

Also during ϕ_3, the state of the one read row that's high is transferred to C_9 in the restore-delay circuit, which stores it long enough to bring up the one corresponding write row line 1 μsec later during the next cycle. The read row actually is inverted twice, by precharging C_{10} at ϕ_1 and then conditionally discharging it at ϕ_2. At the next ϕ_3 pulse, if C_{10} were discharged.

write the first time. To change the contents of the memory, new data is placed on the data feedback path through this feedback circuit, which replaces the feedback stage shown at the right of page 111.
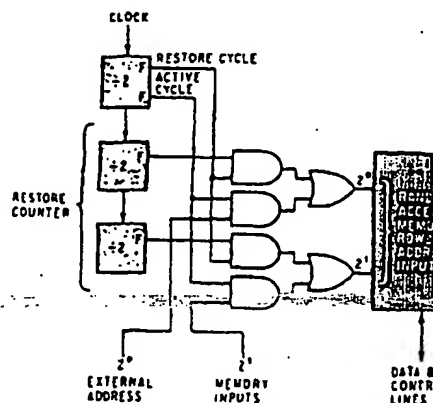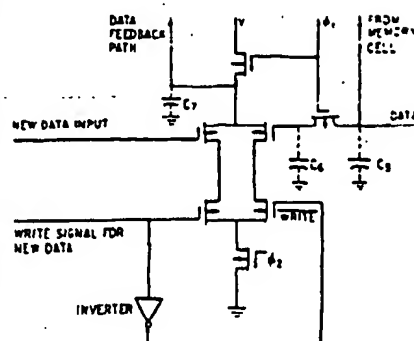
Reading and restoring. Charges on the data-holding capacitors are restored periodically by switching the memory's row inputs to a binary refresh counter. Restore cycles are alternated with command cycles which come to the random-access memory from the computer.
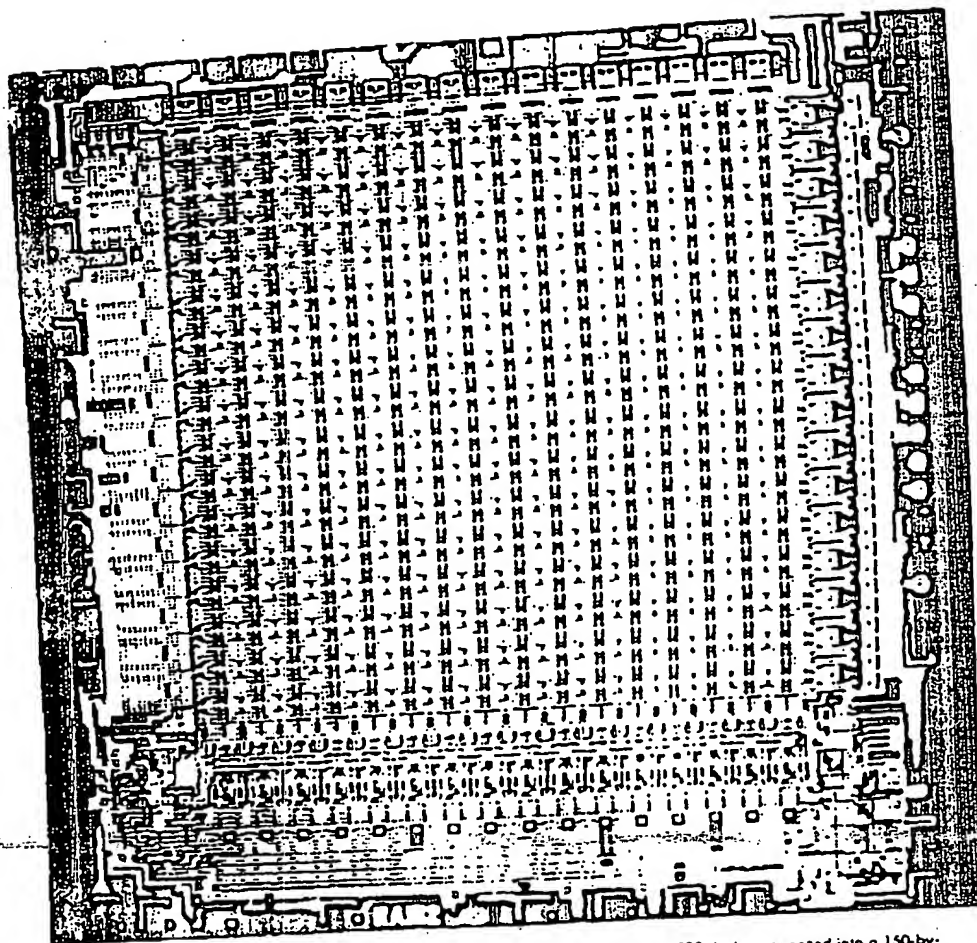
Full chip. Some 4,500 devices squeezed into a 150-by-150-mil-square silicon chip form a complete 1,024, 1-bit-word, four-phase random-access memory. Included on the chip are full binary decoding, chip selection gates, and read-write circuits. 3,072 8-bit-bytes of memory fit on a single 8-by-11-inch printed circuit card (see cover) that includes clock generation and driver circuitry and memory-buffer registers. At a 1-megahertz clock rate, the card dissipates only about 7 watts of power, low enough so that the memory can be driven by a battery in the event of a power failure. Entire computer for which the memory is designed will dissipate only 10 to 15 watts.
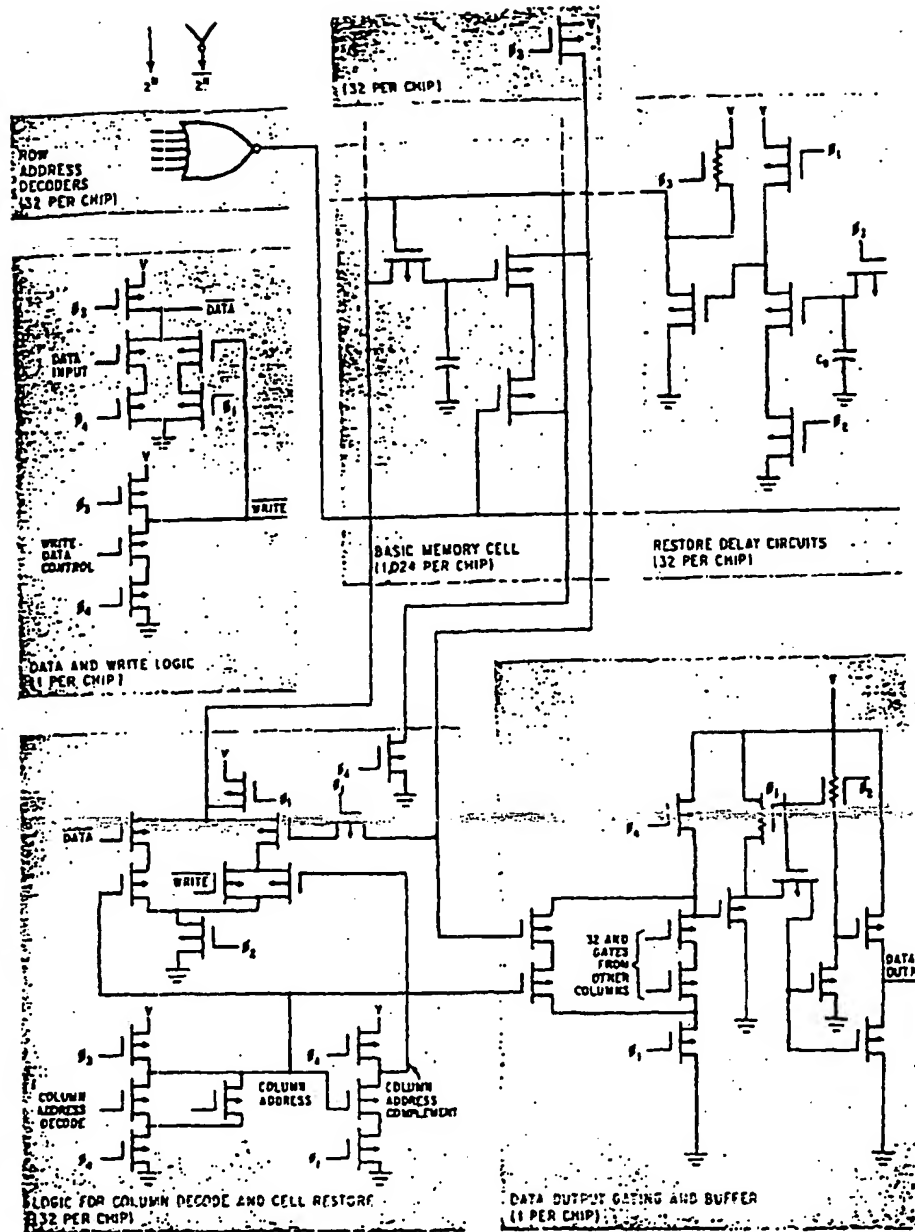
One-chip design. The memory contains all of the circuits required for a 32-row by 32-column random-access memory on a single silicon chip—including one write data input logic stage, one output buffer, and 64 row- and column-selection gates.

E.D. VA. CA 00-524
I 247909

P REGION	
ACTIVE DEVICE OR GATE	
P REGION TO METAL CONTACT	
METAL LINES	

Making the cell. Three-device storage element of the memory occupies 8.9 square mils. Metal row-select and row-refresh lines run horizontally, with the column readout and data feedback lines formed by the vertical p+ regions which act as conductors.
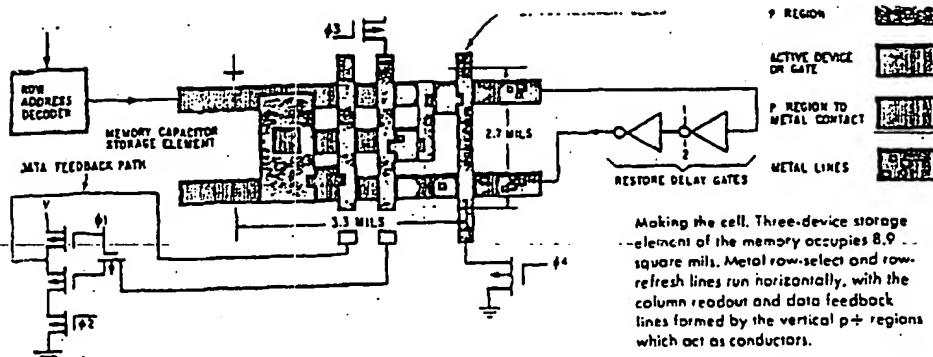
the write row line would rise to restore the charge in the memory cell. If C_{1b} were not discharged the write row line would stay down. The line's level is controlled by the 30:1 ratio of impedances Q_a to Q_b, which form a voltage divider.

This voltage-divider or ratio circuitry depends on the ratio of two MOS impedances; it's an old form of MOS logic. It takes up more room than four-phase logic because the different impedances are obtained by varying the chip area occupied by the individual transistors. However, the ratio circuitry is needed to control the write row lines in spite of the extra space it occupies. This is because the write row lines cannot be controlled by precharging, as are the read row lines, during the four-phase clock times. The memory system would need ber extra clock phases, or additional logic to insure at precharging the write rows would not interfere with other aspects of the memory's operation. The ratio logic approach actually is the simplest way to control the write row line. And there are so few of the circuits on the chip—32 out of thousands—that the extra space required is negligible.

New data may be written into the memory merely by putting it on the feedback path using the circuit shown at the top of page 112. With this circuit, the memory could perform all of the functions of a true random access memory. The integrity of the stored data levels must be assured by interrogating and then refreshing each row at least once every 100 μsec.

To make certain the data will be restored within the allotted 100 μsec each active random-access cycle consisting of the four clock pulses ϕ_1 thru ϕ_4 is followed by a row-restore cycle. Over a long period, data is available from the memory only half the time. It's similar to the situation in a destructive readout core memory, where full cycle time is twice as long as access time—the dead time is needed to rewrite the data that was held in the core location.

In the Four-Phase MOS memory, this dead time—during which the next four clock pulses occur—is used to refresh the data somewhere in the memory. Successive rows are refreshed in successive restore cycles. And these are alternated with random-access cycles.

The addresses of the rows to be refreshed are generated by alternately switching the memory's row inputs between a binary restore counter and the actual address input coming from the computer, as shown on page 112.

The entire memory contains 32 vertical columns, with 32 memory cells in each column. Every column is self-contained and automatically restores its own bits. An entire horizontal row of 32 cells is refreshed during every refresh cycle. This also applies to multiple-chip configurations where chip-select lines are used. Write data input logic and the output buffer appear only once on the chip. There are 32 row address decoder gates, 32 column decoders and 32 restore-delay circuits. Since only rows are refreshed, one five-stage binary counter connected to the row address is all that is needed to cycle through the rows, irrespective of the number of words in the memory.

The memory cell is laid out with the metal row lines running left to right and the p regions, which act like another conductor strip, running up and down, as shown above. Cell size is about 9 square mils, compared to 20 to 30 square mils for a shift register bit. ●

References

1. Lee Boysel, "Cutting system cost with MOS," Electronics, Jan. 20, 1969, p. 105

2. Jack Schmidt, "Integrated MOS Transistor Random Access Memory," Solid State Design, January 1965, p. 21.

3. Jack Schmidt, T. Assi, and J. M. Fridrich, "Hybrid LSI Memory," IEEE Computer Convention, Los Angeles, June 1968.

4. Lee Boysel and Joe Murphy, "100-Nanosecond Memory; 4a Techniques Used in High Speed Logic," EDN, June 1968.

5. Lee Boysel, "Memory on a chip: A step toward LSI," Electronics, Feb. 6, 1967, p. 92.

E.D. VA. CA 00-524
I 247910

February 16, 1970

Electronics

MOS memory: new design makes it big
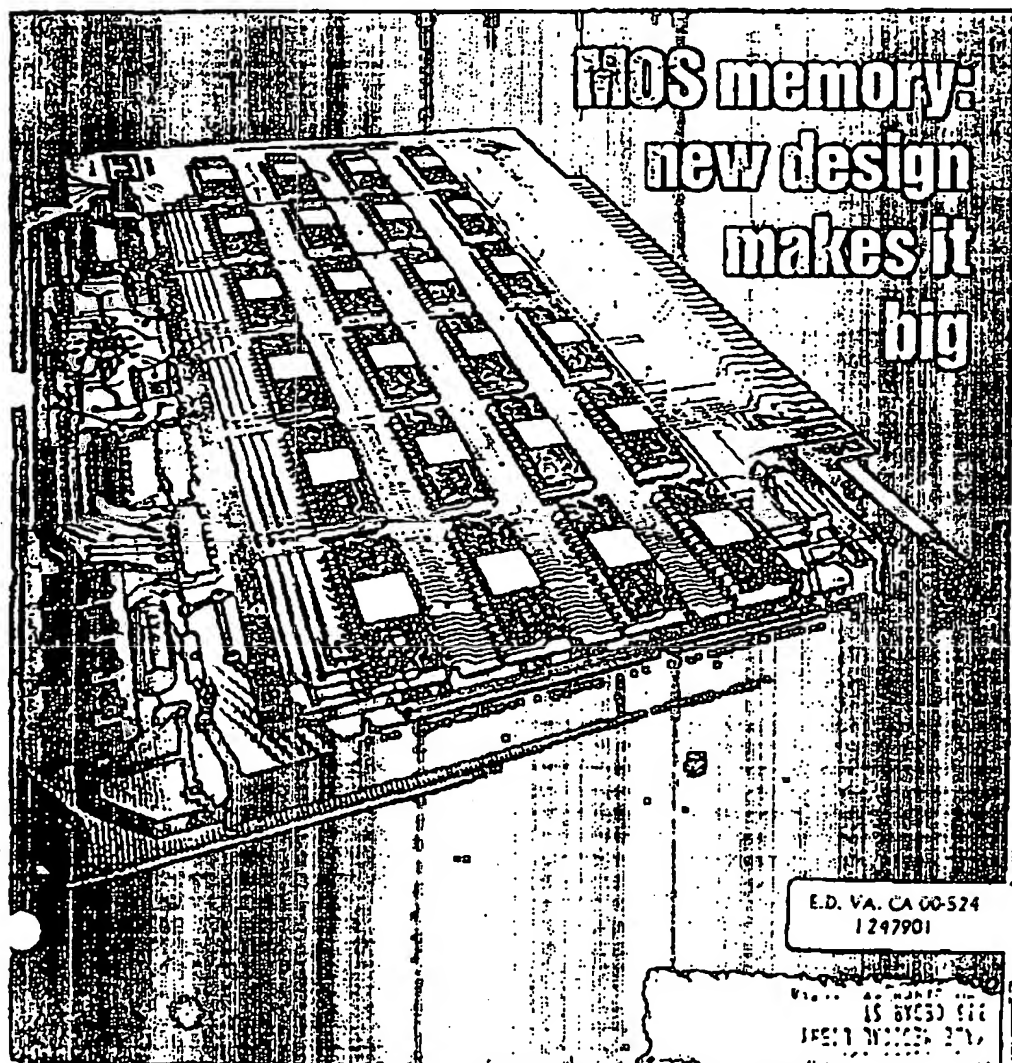
Electronics

Features

Probing the News

U.S. Reports

Electronics International

New Products

Articles

Departments

Who's Who in this issue

Burke   Dutcher

A varied background is one of the strongest assets of Clinton H. Dutcher Jr., author of the article that begins on page 94. A group leader at Electronic Communications Inc.'s R&D division, Dutcher worked on spectral analysis of f-m noise at Bell Labs prior to earning a Ph.D. from Florida University. Co-author Michael R. Burke, who holds a degree from the University of Illinois, now is with Honeywell's Aerospace division.

Moore

Firsts have been a specialty of Gordon E. Moore, author of the article starting on page 126. Under his direction, the Fairchild Semiconductor R&D laboratory scored several firsts, including planar ICs. Moore is now an Intel vice president.

Grueninger

A stickler for selecting the right materials and processes for multilayer circuit boards is Raymond A. Grueninger, who wrote the article starting on page 116. He's with IBM's Electronics Systems Center, and studied chemical engineering.
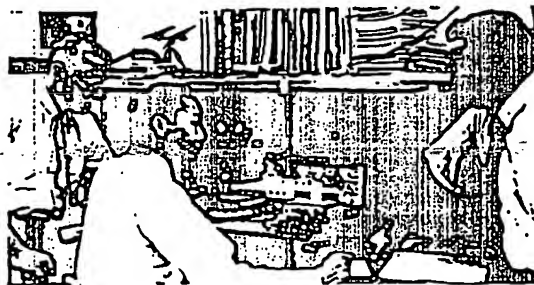
Boysel   Chan   Faith

The art of the possible isn't confined to politics alone, as Lee Boysel, Wallace Chan, and Jack Faith point out in the article beginning on page 109. All three delved into the possibilities inherent in MOS/LSI at Fairchild Semiconductor before moving to Four-Phase Systems. Boysel, president, founded the firm in 1968. Then he recruited Chan, who heads the MOS/LSI design section, and Faith, who is Four-Phase's chief engineer.

Stanley Parnas and Jack Peters, authors of the article beginning on page 101, both are veterans of Sylvania's Applied Research Laboratory. When Synergistics bought the lab in 1969, Peters left to form a consulting firm, while Parnas stayed on. Peters joined Sylvania in 1957; Parnas signed on in 1965.

Arthur Delagrange and Robert Davis, who wrote the article that starts on page 122, met at MIT, where they received MS degrees in electrical engineering. Both men returned to the Naval Ordnance Laboratory, where they had worked as students. Now they work on digital and analog design at NOL.

E.D. VA. CA 00-524
1 247903
9

A RISC MICROPROCESSOR WITH INTEGRAL MMU AND CACHE INTERFACE

Craig Hansen, Dan Freitas, Ed Hudson,
John Moussouris, Steve Przybylski, Tom Riordan, and Chris Rowen

MIPS Computer Systems
930 Arques Ave.
Sunnyvale, CA 94086

Abstract

Unique memory and coprocessor interfaces sustain bandwidth greater than 128 Mbyte/sec between a single-chip RISC processor and external caches of up to 128 Kbytes of standard 25-35 ns CMOS static RAM. Chip crossings are minimized by integrating all cache control and virtual memory circuitry on chip, including tag comparators, parity generators and checkers, refill buffers, TLB (Translation Lookaside Buffer), clock generators, and bus control logic. Novel circuit techniques reduce translation time, inductive switching noise, and clock skews.

Introduction

The MIPS R2000 is a 32-bit CMOS RISC processor that executes all instructions using a single cycle of a five-stage pipeline. The instruction set is reduced down to functions that are performance-critical, universal, and well-matched to both the hardware pipeline and the compiler system. As a result, the processor itself consumes only about one-third of the area of this 85 mm² chip.

The R2000 processor is capable of very fast execution of compiled programs, but sustaining this performance level over a range of applications in a multi-tasking environment requires the efficient support of virtual addressing and operating systems. High execution rate requires a high memory system bandwidth and low-latency memory operations. To address these concerns, much of the remaining two-thirds of the area, freed up by the reduction of complexity of the processor itself, is devoted to an on-chip virtual memory system and a high-bandwidth cache interface.

Virtual Memory

The virtual memory system employs a 64-entry fully-associative TLB to translate virtual addresses to physical locations. A 6-bit process identifier appended to virtual addresses permits fast context switching without TLB flushing. Hardware registers provide pseudo-random addressing of 56 of the 64 entries for fast TLB refill; the remaining 8 entries are directly addressable, so that certain virtual pages can be locked into the TLB. Instruction and data addresses are always translated before accessing the caches and memory system; in particular, both the instruction and data caches are physically addressed.

TLB entries may be read or written by explicit software control, but the TLB has no direct access to memory for refill on misses; instead, a TLB miss provides transfer of control to a software routine, which fetches a page table entry and writes it into the TLB. For a conventional one-level page table, special registers provided in the system coprocessor streamline this routine to a minimum of 9 instructions, averaging about 20 cycles, including cycles for cache misses and exception-handling latency. By changing this software routine, support for segmented or object-oriented virtual addressing can be provided as well.

A fast 2-entry micro-TLB accelerates instruction translations for conditional branch operations. This micro-TLB is transparently filled from the TLB in a single cycle. Because it is small, no process identifier need be matched in the micro-TLB; instead, it is transparently flushed when the process identifier changes.

In addition to performing address translation, the TLB also provides control of caching and accessibility of virtual pages. Uncached pages are typically used for reference to I/O devices, to perform copying of data without flushing the cache, and to bootstrap the processor before initializing the cache. Virtual pages may be marked as global, and matched without regard to the process identifier, thereby supporting shared, global, virtual addressing for use in shared code libraries or other applications.

Cache Interface

Figure 1 illustrates the three major busses and control interfaces of the processor. The ADDRESS bus transmits the lower 16 bits of physical address to latching buffers that drive two external cache arrays for instructions and data respectively on alternate 30 ns phases. The DATA bus transfers 32 bits of instructions or operands (plus 4 parity bits) during the following phase, so that each direct-mapped cache cycles in 60 ns. The TAG bus transfers the upper 20 bits of physical address (plus 3 parity and a valid bit).

Cache sizes of 4 Kbytes to 64 Kbytes are accommodated by redundantly supplying 4 bits of the physical address to both the ADDRESS bus and TAG bus. DATA and TAG both drive onto the processor chip during fetches of instructions or data, and off chip during stores to data cache and main memory.

Standard static RAM devices are used for the cache memory. The processor generates and checks parity on all transactions with the cache memories to assure data integrity. The parity checking circuits may themselves be checked by forcing zero values from the parity generation circuits. Diagnostic facilities permit the explicit loading, checking, and flushing of the data cache without causing implicit memory operations. The bus connections of the instruction and data caches are completely symmetric, and so by interchanging the cache control signals, the two caches are swapped thereby providing diagnosis of the instruction cache as well.[1]

All tag checking is performed within the processor, and when cache misses or parity errors are detected, memory reads are performed to refill the cache and resume execution. Memory writes are initiated in parallel with data cache writes, and are externally buffered.

Storage Interface

The storage control interface on the right side of Figure 1 includes signals for initiating memory reads and writes with various byte access types, interlocking on busy conditions, and responding to bus error and up to 6 external interrupt types. A late retry/bus error mechanism provides for operation with error-correcting memory systems without impacting access time. A variety of memory systems can be connected to the storage interface, providing a wide range of system cost/performance levels.[2]

Coprocessor Interface

The coprocessor interface on the left side includes signals for asserting and interlocking on coprocessor conditions, and for synchronizing pipelines in the presence of stalls and exceptions. The processor provides instructions that load and store words to the coprocessor, with all addressing and cache refills handled by the processor. Normally, the coprocessor has a separate register file for holding operands. This synchronous interface supplies instructions and data at the 128 Mbyte/sec data bus rate to external coprocessor units for floating point and other special functions.

Block Diagram

Figure 2 is a block diagram and floor plan of the processor chip, which consists of two tightly-coupled units.[3] On the right side is the 32-bit RISC CPU that executes the simple loads, stores, branches, and register-to-register operations most frequently required by optimizing compilers[4] at a rate of one instruction per 60 ns cycle. This CPU includes a 32x32 register file, load aligner, 32-bit ALU and shifter, an autonomous multiply/divide unit, and an address unit that generates 32-bit virtual addresses for data and instructions on alternate 30 ns phases. On the left side is the System Coprocessor that includes the 30 ns 64-entry fully-associative TLB, 5 special registers for TLB refill and other memory management functions, and 3 special registers for diagnostics, error recovery, and exception-handling support of a multi-tasking operating system.[5]

Pipeline Organization

Figure 3 details how the blocks described are fit together in a five-stage pipeline. The instruction and data caches, ICACHE, DCACHE, are each permitted a full cycle to operate, offset a half cycle apart so that they can both occupy the same buses without interference. This offset is well-matched to the address generation path, which uses three half-cycles each to perform register file access, RF, virtual address generation, DVA, and virtual address translation, DTLB. Register file reads, RF, and writes, RW, are fully bypassed, so that successive ALU operations can be performed back-to-back, with load operations adding a single additional cycle of latency. Branches have similar latency, due to the micro-TLB and a reduction of the instruction set to simple branch conditions.

Circuits

Special circuit design techniques are employed in the R2000 to achieve tight control of timing skews. Figure 4 shows the TLB circuits. The match lines are precharged high. When the dummy match line (driven by the dummy data line) is sensed low, a strobe is sent to all other match output latches. Timing on the dummy path actually precedes the normal data path by the dummy match sense/strobe driver time. These self-timed clocking techniques are insensitive to process variation, and help to achieve a translation time of 20 ns.

Similar balanced-path techniques are used in the clock circuitry to obtain precise control of sample strobe, address setup, read control pulse, and tristate output timings. Four classes of timing events are generated from four 32 MHz clock inputs that pass through identical divide-by-two buffers to achieve a wide range of tolerance to input duty cycle. Skew is controlled externally by adjusting the relative phases of the four clock inputs. Since all on-chip clock levels and paths are identical, relative delays are stable under process variation.

The processor dissipates less than 3 W in a 144-pin ceramic package with 109 TTL-compatible signal I/Os and 35 power and ground pins. Timings described in this paper are achieved at worst-case conditions of 145 C junction temperature and 4.2 V power supply. At a 16.67 MHz peak instruction rate, this RISC processor, supported by an appropriate memory hierarchy, can sustain performance of about 10 times a VAX† 11/780 on UNIX‡ system benchmarks.

The chip is fabricated in a conservative double-metal single-poly CMOS technology, with a 2 micron drawn channel length and 400 angstrom gate oxide. The 8.5 x 10mm die contains approximately 100,000 transistors. Full-custom layout follows a twin-tub methodology with conservative latchup protection, for robustness in transporting the design to faster technologies.

Conclusion

The MIPS R2000 processor relies upon integrated virtual addressing support and cache interfaces to provide short-latency load, store and branch instructions that maintain the inherent performance advantage of RISC processors in a full system environment. The processor permits large split caches to be constructed at low cost, using standard static RAM devices. Flexible coprocessor and memory interfaces allow the basic design to be extended with additional instructions and high-performance memory systems that provide headroom for systems of even greater speed. Diagnostic facilities provide highly reliable and testable system environments.

1. Robert H. Abramovitz and Bob Patrie, "Design for Testability in a RISC Environment," *Proceedings ICCD*, IEEE, (October 6-9, 1986).

2. Phil Gossett, Marvin Mills, Prabhat Misbra, Wally Pearson, and Skip Stritter, "Engineering a RISC Hardware Environment," *Proceedings ICCD*, IEEE, (October 6-9, 1986).

3. J. Moussouris, L. Crudele, D. Freitas, C. Hansen, E. Hudson, S. Przybylski, T. Riordan, and C. Rowen, "A CMOS RISC Processor with Integrated System Functions," *Proceedings 1986 COMPCON*, IEEE, (March 4-6, 1986).

4. Fred Chow, Mark Himelstein, Earl Killian, and Larry Weber, "Engineering a RISC Compiler," *Proceedings 1986 COMPCON*, IEEE. (March 4-6, 1986).

5. M. DeMoney, J. Moore, and J. Mashey, "Operating System Support on a RISC," *Proceedings 1986 COMPCON*, IEEE, (March 4-6, 1986).

147
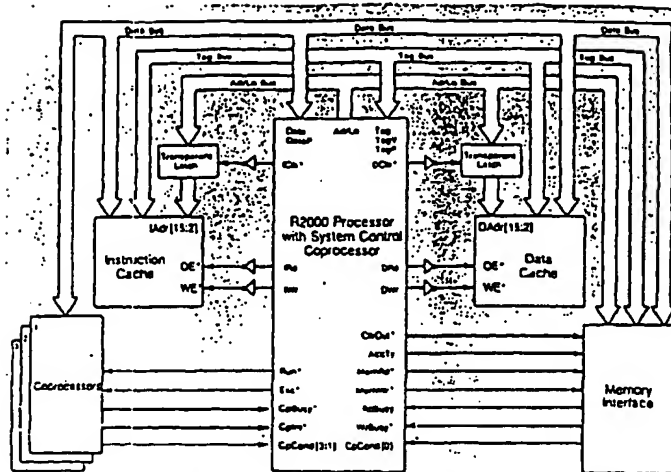
Figure 1. R2000 System Block Diagram
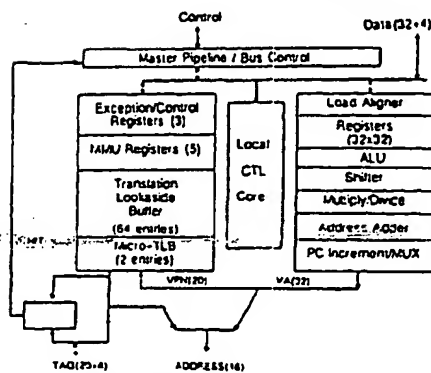
Figure 2. R2000 Block Diagram

Figure 3. R2000 Pipeline

ICACHE	Instruction cache access
DCACHE	Data cache access
IVA	Instruction virtual address generation
DVA	Data virtual address generation
ITLB	Instruction translation
DTLB	Data translation
IDEC	Instruction decode
RF	Register fetch
OP	Operation
RW	Register writeback
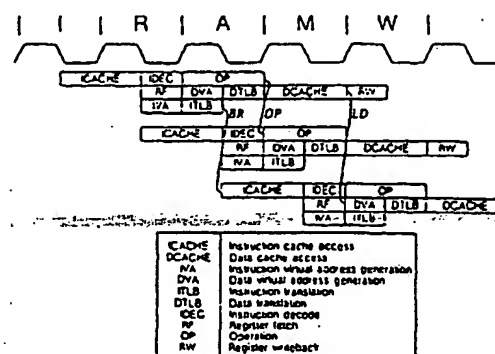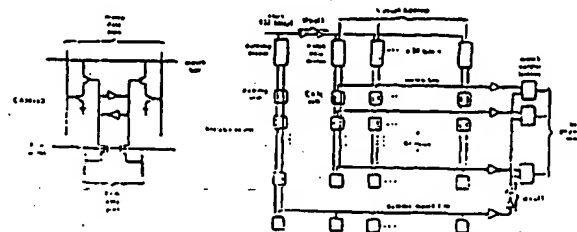
Figure 4. TLB Circuits

148

A CMOS RISC PROCESSOR WITH INTEGRATED SYSTEM FUNCTIONS

J. Moussouris, L. Crudele, D. Freitas, C. Hansen, E. Hudson
R. March, S. Przybylski, T. Riordan, C. Rowen, D. Van't Hof

MIPS Computer Systems
930 Arques Ave.
Sunnyvale, CA 94086

Abstract

By omitting complex features and streamlining the most frequent operations, reduced instruction set computers (RISC) can achieve high peak performance at relatively low hardware cost, particularly when implemented in VLSI.[1,2] But to sustain this performance across a broad range of environments, integration of critical system functions is as essential as reduction of the instruction set. We describe a single-chip CMOS processor that consists of a very lean RISC CPU that achieves 16 mips peak, along with a system coprocessor that integrates the functions needed to keep the CPU from idling for storage access.

Introduction

MIPS Computer Systems of Sunnyvale, Ca. has developed a full-custom CMOS VLSI processor consisting of two tightly-coupled 16 MHz units on a single chip. The first unit is a RISC CPU that has powerful data handling facilities, but is simpler at the machine code / compiler interface than most other RISC processors.

The second unit is a system coprocessor that is designed to fit the needs of a multi-tasking operating system for virtual memory, exception handling, and error recovery. This coprocessor incorporates a memory/cache interface with on-chip tag comparators, parity generators and checkers. It generates the clock and control signals needed to achieve peak bus bandwidths of more than 128 Mbyte/sec with external caches of up to 128 Kbytes of standard 25-35 nsec CMOS static RAMS. Finally, it supports a synchronous coprocessor interface that supplies this bandwidth to external coexecution units for floating point and other special compute-intensive functions.

The MIPS processor was developed jointly with an optimizing compiler suite and UNIX* OS port. The

* UNIX is a Trademark of AT&T.
VAX is a Trademark of Digital Equipment Corp.

compiler was bootstrapped and tested, and the critical paths through the OS were running on a cross-assembler, well before hardware design was complete. Hence design decisions could be made in the presence of quantitative measurements of impact on overall system performance. Many tradeoffs were explored. Some innovations not only enhanced performance, but also simplified life for the VLSI and system designers, and for the OS and compiler developers as well! The final result is a 16 MHz TTL-compatible CMOS chip which dissipates less than 2 W in a 144-pin ceramic PGA package, and provides sustained performance about 8 times a VAX* 11/780 across a broad range of application and system programming environments.

This paper describes the hardware: what was left out and what was put into the VLSI implementation. Two companion papers discuss some of the tradeoffs and innovations in compiler[3] and OS.[4]

What we left out

The MIPS instruction set is designed to execute effectively in a single cycle, in a deep synchronous pipeline, interruptible on cycle boundaries. There is no microcode. As in other RISC machines, all computation is register-to-register, and all data accesses are simple loads and stores.[5]

The MIPS architecture is, however, even simpler and leaner than most other RISC machines. The hard-wired machine code is free of factors that could degrade cycle time, pipeline efficiency, or responsiveness and precision of the exception mechanism. After extensive performance analysis, a number of features that are common even in RISC machines were left out, including the following:

Hidden registers. Figure 1 illustrates the MIPS CPU registers: 32 general-purpose 32-bit registers, a double-word (64-bit) special register for multiply and divide results, and a 32-bit program counter. The general-purpose registers are all directly and simultaneously addressable from every instruction. There are no mechanisms for hiding a portion of the

126

General Purpose Registers

	r0
	r1
	r2
	:
	r29
	r30
	r31

Multiply / Divide registers

	HI

	LO

Program Counter

	PC

Figure 1 - CPU Registers

I-type (immediate)

op	rs	rt	immediate

J-type (jump)

op	target

R-type (register-to-register)

op	rs	rt	rd	sh	function
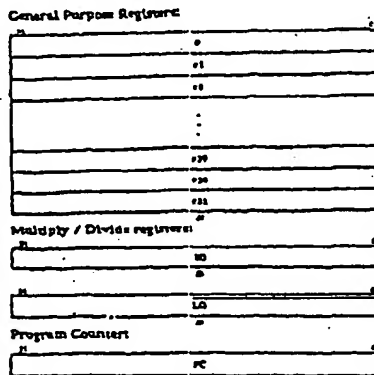
Figure 2 - Instruction Formats

register file, such as the following schemes implemented in other RISC machines: register windows,[6] stack caches,[7] separate user/kernel registers,[8] or process register sets.[9] Instead, the register file is symmetric, and the compiler and OS implement various software strategies (some of which are described in the two companion papers), for dramatically reducing the overhead of saving and restoring registers. The cache and memory controller also assist by buffering successive storage operations.

Condition codes. In the MIPS architecture, conditions generated by SET instructions are loaded directly into the general-purpose registers (except for overflow, which is trapped). There is no condition code register. Hence the pipeline design is freed from any special mechanisms to bypass condition codes, interlock on them, or abort writing them on exceptions — beyond those implemented for the register file itself. Moreover, conditions mapped onto the register file are subject to the same compile-time optimizations in allocation and reuse as other register variables.[10]

Variable-length instructions. Figure 2 illustrates the three CPU instruction formats, all of which are fixed 32-bit words. Simple instruction decoding guarantees 100% utilization of the instruction cache bandwidth, without prefetch buffers and

complications in the exception model caused by having instructions straddle page boundaries. The effect of long instructions can be synthesized at compile time. For example, 32-bit immediate addresses are synthesized by concatenating two I-type instructions (see Figure 2) containing the upper and lower 16-bits of the immediate respectively.

Multiple address modes. All MIPS loads and stores are I-type instructions, implementing a single (base + 16-bit offset) address mode. Similarly, all branches are I-type with a single (PC + 16-bit offset) address mode, and all jumps are either J-type with a 26-bit absolute word address or R-type with a 32-bit register target. In each case, the mode implemented is the one most frequently needed by compilers. Complex modes, such as (base + index + offset), are synthesized at compile time, subject to optimizations that eliminate redundancy.[11] Implementing only the frequent mode in hardware minimizes register ports, datapath busses, and pipeline latencies for loads and branches.

What we put in

Omission of complicated features from the MIPS machine code makes available silicon area and power for data handling, concurrency, and system functions that substantially enhance the performance and versatility of the processor. As before, we emphasize how the MIPS design differs in these areas from other RISC machines.

Data Handling

Since loads and stores have only one address mode, a lot of opcode space is available for multiple data types. MIPS supports signed and unsigned loads and stores of bytes, halfword, and full words. In

127

addition, there are some special data handling features:

Unaligned reference support. MIPS defines instructions for accessing unaligned words and halfwords. For example, LWR (LWL) extracts the right (left) fragment of an unaligned word from a given aligned word in memory, and right (left) justifies it into a designated general purpose register. Two of these instructions can be concatenated to perform a general unaligned word reference in the minimal two cycles (the load aligner is bypassed internally to merge the fragments).

Dual byte sex. The MIPS processor has two byte sex configurations: little-endian (VAX, x86, 32x) and big-endian (370, 68x). Hence it is compatible with existing databases generated by machines that access bytes in either order.

Concurrency

The MIPS machine achieves single-cycle execution of its simplified instructions, including loads and branches, thanks to the concurrency achieved in an efficient pipeline and multiple functional units.

Single cycle loads and branches. MIPS loads and branches execute in precisely one cycle, with a single additional cycle of latency. There are no restrictions on concatenating storage instructions back-to-back, and no hardware interlocks.[13] Instead, loads and branches always take effect just after the instruction that follows them (the "delay slot"). The MIPS assembler reorders instructions to fill delay slots with useful code 70-90% of the time.

Efficient pipeline. Figure 3 illustrates four successive instructions in the MIPS pipeline. The 16.6 MHz clock cycle is divided into two 30 nsec phases. Note that the external instruction and data caches each have 60 nsec to cycle. The major internal operations (OP, DA, IA) each occur in 60 nsec as well. Instruction decode is simple enough, however, to occur in a single 30 nsec phase, overlapped with register fetch. Calculation of a branch target (IA) also overlaps IDEC, so that a branch at instruction 0 can address the ICACHE access of instruction 2 (see dotted line A in figure 3). Similarly, a load at instruction 0 gets its data bypassed into the OP of instruction 2 (dotted line C), but an ALU/shift result gets bypassed directly into instruction 1 with zero latency (dotted line B).
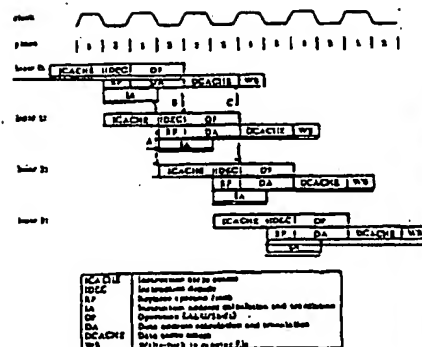
Figure 3 - Pipeline

Note that the IA-ICACHE and DA-DCACHE cycles are displaced by one phase, so that the corresponding TLB and cache accesses can be interleaved on a single set of busses.

Multiple functional units. Figure 4 is a block diagram of the MIPS processor. On the right side is the CPU datapath that implements the pipeline of Figure 3. There is a stack of functional units, including ALU, 32-bit shifter, and an autonomous 32-bit multiply/divide unit. An address adder and incrementer/mux for the PC generate data and instruction virtual addresses alternately at 30 nsec intervals. After a 60 nsec latency, operands or instructions are transferred (again, on alternate phases) across the external DATA bus. Instructions are latched into a central local decode core, and also into a master pipeline bus controller that coordinates internal and external pipeline events in the presence of stalls and exceptions.

The datapath is organized so that all units can initiate their functions under local control. When instruction decode is complete, the master control unit can then late-select the desired results at the destination point, and abort any unused functions.

Referring again to Figure 3, we see that during a typical phase 1, instruction 3 might be bringing back an instruction onto the DATA bus from ICACHE, 2 might calculate a data address, 1 might initiate a DCACHE access, and 0 might be writing back the result of a previous load to the register
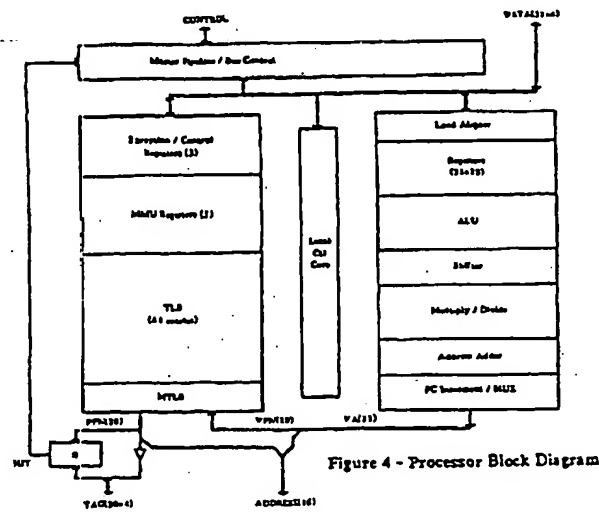
Figure 4 - Processor Block Diagram
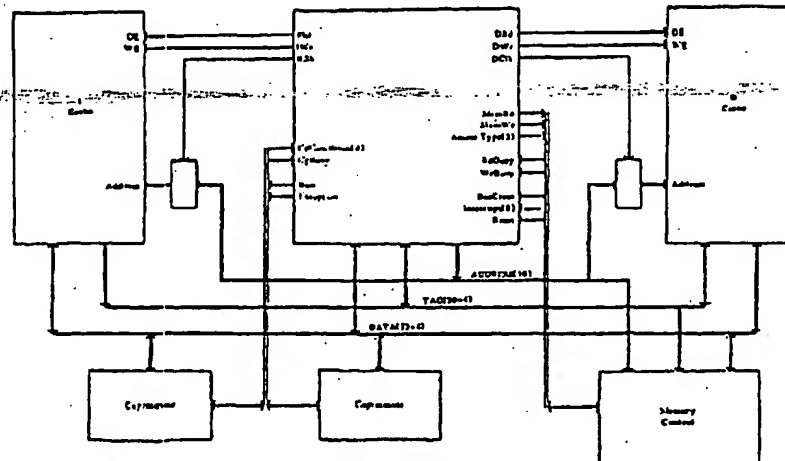
Figure 5 - Interface

129

Ate. Meanwhile, a multiply or divide operation that was initiated during a still earlier cycle might be preparing a 64-bit result to load into the HI/LO registers.

Integrated system coprocessor

On the left side of the Figure 4 block diagram is the datapath of the system coprocessor, which implements yet another level of concurrent functions as follows:

Virtual memory. The major element in the system coprocessor data path is a 64-entry fully associative translation lookaside buffer (TLB), which translates a 20-bit virtual page number into a 20-bit physical page frame number in a 30 nsec phase. Instruction addresses translate in a faster 2-entry micro-TLB (MTLB), to compensate for branch target calculation exceeding 30 nsec. When the MTLB misses, it is filled from the main TLB in a single cycle — a small penalty, since accesses stay within the two most recent pages 95-99% of the time. When the TLB itself misses, 5 special-purpose MMU registers and 3 exception/control registers in the coprocessor assist the operating system in refilling from an external page table, using a TLB miss handler with fewer than 10 instructions for the most frequent case. The format and function of the TLB and system coprocessor registers are detailed in the OS paper in this series.[4]

Exception handling and error recovery. The MIPS machine supports a rich set of exception types, including address and translation exceptions, and illegal operation, overflow, and program traps. External asynchronous exceptions include six maskable external interrupts, bus error, and reset.

All exceptions can occur at cycle boundaries. The exception model is precise. That is, each exception is handled in a state that reflects serial completion of all instructions prior to the exception and none of the instructions subsequent to it. The simple CPU register set (Figure 1), reduced instruction set (Figure 2), and synchronous contention-free pipelining (Figure 3) play a key role in flushing and restarting the pipe gracefully to maintain precise exceptions with only 60 nsec latency.

The system Status register contains fields which assist in error diagnosis and recovery. Further details of the exception-handling and error-recovery mechanisms are given in the OS paper in this series.

External cache interface. As shown in Figure 4, the physical address coming out of the TLB is split across two external busses: ADDRESS low (16 bits) and TAG (20 bits, plus valid and 3 parity bits). The system coprocessor includes on-chip parity checkers and generators, as well as a tag comparator that detects cache HIT.

Figure 5 illustrates the external interface of the MIPS processor. Two external arrays of standard CMOS SRAMs are organized as direct-mapped split instruction and data caches. The processor interleaves accesses to the two caches on the ADDRESS, TAG, and DATA busses. Instruction fetch begins with ADDRESS clocked through a latching buffer by ICLK during phase 2, and continues until DATA and TAG are latched on the chip at the end of the next phase 1. Similarly, data fetch begins with ADDRESS clocked by DCLK during phase 1, and completes with DATA and TAG latched on the chip at the end of phase 2. During data stores, all three busses are outputs from the chip to the memory controller: the full 32-bit real address is transmitted on ADDRESS and TAG during phase 1, and the 32-bit DATA during phase 2.

This cache interface integrates all circuitry that normally intervenes between processor and raw cache RAMs. Even the control lines for cache write and tristate output enables and for the address buffer clocks are all generated on chip, for precision in control of clock skews. Without this level of integration it would not be feasible to sustain bus bandwidth in excess of 128 Mbyte/sec with standard 25-35 nsec CMOS SRAMs. The OS also plays a role in managing the caches to maintain consistency with DMA I/O activity without hardware bus watching overhead.

Memory interface. The memory control interface on the right side of Figure 5 includes several signals that synchronize storage events. MemRd and MemWr are asserted on cache miss and store respectively, while AccessType distinguishes null, byte, halfword, tribyte, and word transfers. RdBusy and WrBusy control termination and initiation of the stalls that occur when the cache misses or the write buffer is full. Bus Error warns of hard storage errors, such as non-recoverable ECC conditions or bus timeout. Other asynchronous events are signalled on six external interrupt lines. The memory interface can also support configurations with one or both caches missing.

External coprocessor interface. Figure 5 also illustrates the external coprocessor interface, which is designed to support a floating point unit,

Table of Contents

ix

vi

xii

xiii

xiv

LSI LOGIC

LR2000
High Performance
RISC Microprocessor
Preliminary

Description

The LR2000 CPU is a high-speed HCMOS implementation of the MIPS RISC (Reduced Instruction Set Computer) microprocessor architecture. The MIPS architecture was initially developed at Stanford University under the auspices of DARPA. The LR2000 is an extension of the Stanford MIPS architecture developed by MIPS Computer Systems, Inc. This architecture makes possible a microprocessor that can execute instructions for high-level language programs at rates approaching one instruction per processor clock. It supports up to three tightly coupled coprocessors including the single chip LR2010 Floating-Point Accelerator.

The full-custom 32-bit VLSI CMOS Reduced Instruction Set Computer shown in the CPU chip photo includes thirty-two 32-bit registers, on-chip TLB (translation lookaside buffer), memory management unit, and cache control circuitry.
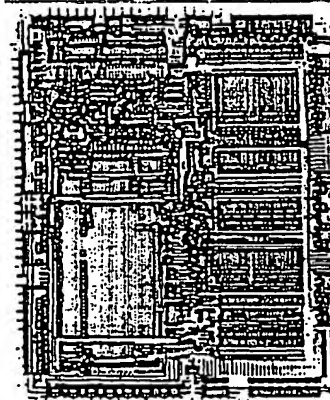
LR2000 CPU Chip Photo

Features

- Reduced Instruction Set Computer (RISC) architecture
  - MIPS instruction set
  - Simple 32-bit instructions, single addressing mode
  - Register-to-register, load-store operation
  - All instructions (except MPY and DIV) execute in a single cycle
- High performance
  - Fast instruction cycle with five-stage pipeline
  - Efficient handling of pipeline stalls and exceptional events
- Two speed versions
  - LR2000/12 12.5 MHz 8 VAX mips equivalents
  - LR2000/16 16.7 MHz 10 VAX mips equivalents
- Optional devices tightly coupled for high performance
  - LR2010   Floating-Point Accelerator (FPA)
  - LR2020   Write Buffers (WB)
- 32 general-purpose registers
- On-chip cache control
  - Separate external instruction and data cache memories
  - From 4 to 64 Kbytes each

- Both cache memories accessed during a single CPU cycle
- Dual cache bandwidth up to 133 Mbytes/second
- Uses standard SRAMs
  - LR2000/12 35 ns access time
  - LR2000/16 25 ns access time
- On-chip memory management unit (MMU)
  - Fully-associative, 64-entry translation lookaside buffer (TLB)
  - Supports 4-Gbyte virtual address space
- Multi-tasking support
  - User and kernel (supervisor) modes
- Seamless coprocessor interface
  - Generates all addresses and handles memory interface control
  - Supports up to three external coprocessors
- Strong, integrated software support
  - UMIPS operating system
    - System V.3,4.3 BSD
  - Optimizing compilers
    - C   Ada   (Verdix)
    - FORTRAN   COBOL(LPI)
    - Pascal   PL-1   (LPI)
- 144 ceramic pin grid array package

MR0105703

**LR2000
High Performance
RISC Microprocessor
Preliminary**

Introduction

The LR2000 processor consists of two processors implemented on a single chip. In addition to a RISC (Reduced Instruction Set Computer) CPU there is a system control coprocessor (CP0) which contains a TLB and control registers to support a virtual memory subsystem.

Block Diagram
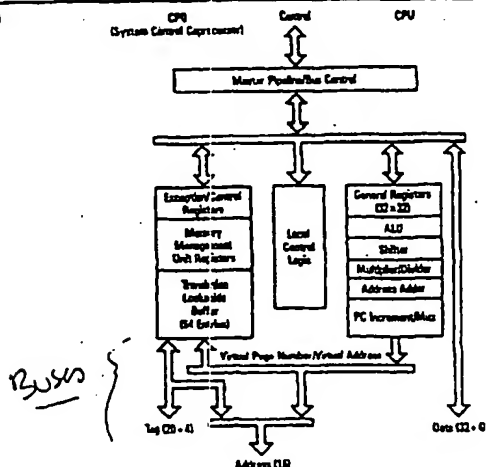
Instructions

All LR2000 instructions are 32 bits in length. To simplify instruction decoding, only three instruction formats are supported (immediate, jump and register). The instruction set can be divided into the following groups.

■ Load/Store instructions move data between memory and the general registers. All instructions are then executed on values stored in the general registers. There are no operations performed on operands in cache or main memory. Loads and stores are all I-type instructions since the only addressing mode supported is base register + 16-bit immediate offset.

■ Computational instructions perform arithmetic, logical and shift operations on values in registers. These can be R-type (both operands are registers) or I-type (one operand is a 16-bit immediate) instruction formats.

■ Jump and Branch instructions change program flow. Jumps are always to an absolute 26-bit address (J-type format for subroutine calls) or 32-bit register byte addresses (R-type for returns and dispatches). Branches have 16-bit offsets relative to the program counter (I-type). Jump and link instructions save a return address in register 31.

■ Coprocessor instructions perform operations in the coprocessors. Coprocessor loads and stores are I-type, or have coprocessor-dependent formats. Coprocessor 0 instructions perform operations on the CP0 registers to manipulate memory management and exception handling facilities.

■ Special instructions perform a variety of tasks including movement of data between special and general registers, trap and breakpoint. They are always R-type.

The LR2000 CPU provides 32 general purpose 32-bit registers, a 32-bit program counter and two 32-bit registers which hold the results of integer multiply and divide operations. The functions traditionally provided by a program status word (PSW) register are handled by the status and cause registers in the CP0.

The LR2000 supports a user and kernel (supervisor) mode. The LR2000 normally operates in user mode until an exception is detected forcing it into the kernel mode. It remains in kernel mode until a restore from exception (RFE) instruction is executed. The 4-Gbyte address space is divided into 2 Gbytes for users and 2 Gbytes for the kernel.

2

MR0105704

LR2000
High Performance
RISC Microprocessor
Preliminary
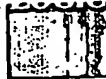
— not performed in cache —
nonreadable memory

Table 1. Instruction Summary

OP	Description
	Load/Store Instructions
LB	Load Byte
LBU	Load Byte Unsigned
LH	Load Halfword
LHU	Load Halfword Unsigned
LW	Load Word
LWL	Load Word Left
LWR	Load Word Right
SB	Store Byte
SH	Store Halfword
SW	Store Word
SWL	Store Word Left
SWR	Store Word Right
	Arithmetic Instructions (ALU Immediate)
ADDI	Add Immediate
ADDIU	Add Immediate Unsigned
SLTI	Set on Less than Immediate
SLTIU	Set on Less than Immediate Unsigned
ANDI	AND Immediate
ORI	OR Immediate
XORI	Exclusive OR Immediate
LUI	Load Upper Immediate
	Arithmetic Instructions (3-operand, register-type)
ADD	Add
ADDU	Add Unsigned
SUB	Subtract
SUBU	Subtract Unsigned
SLT	Set on Less than
SLTU	Set on Less than Unsigned
AND	AND
OR	OR
XOR	Exclusive OR
NOR	NOR
	Shift Instructions
SLL	Shift Left Logical
SRL	Shift Right Logical
SRA	Shift Right Arithmetic
SLLV	Shift Left Logical Variable
SRLV	Shift Right Logical Variable
SRAV	Shift Right Arithmetic Variable

OP	Description
	Multiply/Divide Instructions
MULT	Multiply
MULTU	Multiply Unsigned
DIV	Divide
DIVU	Divide Unsigned
MFHI	Move From HI
MTHI	Move to HI
MFLO	Move From LO
MTLO	Move to LO
	Jump and Branch Instructions
J	Jump
JAL	Jump and Link
JR	Jump to Register
JALR	Jump and Link Register
BEQ	Branch on Equal
BNE	Branch on Not Equal
BLEZ	Branch on Less than or Equal to Zero
BGTZ	Branch on Greater than Zero
BLTZ	Branch on Less than Zero
BGEZ	Branch on Greater than or Equal to Zero
BLTZAL	Branch on Less than Zero and Link
BGEZAL	Branch on Greater than or Equal to Zero and Link
	Special Instructions
SYSCALL	System Call
BREAK	Break
	Coprocessor Instructions
LWCz	Load Word to Coprocessor
SWCz	Store Word from Coprocessor
MTCz	Move to Coprocessor
MFCz	Move from Coprocessor
CTCz	Move Control to Coprocessor
CFCz	Move Control from Coprocessor
COPz	Coprocessor Operation
BCzT	Branch on Coprocessor z True
BCzF	Branch on Coprocessor z False
	System Control Coprocessor (CP0) Instructions
MTC0	Move to CP0
MFC0	Move from CP0
TLBR	Read Indexed TLB Entry
TLBWI	Write Indexed TLB Entry
TLBWR	Write Random TLB Entry
TLBP	Probe TLB for Matching Entry
RFE	Restore from Exception

3

MR0105705

LR2000
High Performance
RISC Microprocessor
Preliminary

R2000 Register rganization

32	General Purpose Registers	0
	r0 Hardwired 0	
	r1	
	r2	
	•	
	•	
	•	
	r29	
	r30	
	r31	

32	Multiply/Divide Registers	0
	HI	
	LO	

32	Program Counter	0
	PC	
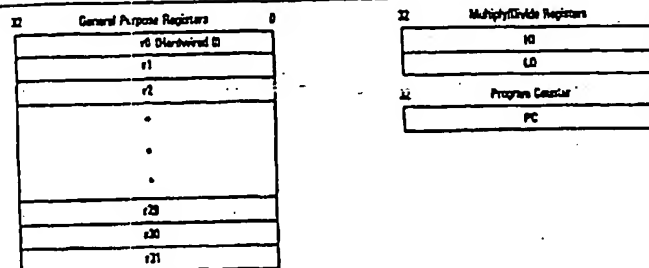
Figure 1. CPU Registers

System Control Coprocessor (CP0)

The LR2000 can operate with up to four tightly coupled coprocessors (CP0 thru CP3). The system control coprocessor (CP0) is on the LR2000 chip and supports the virtual memory system and exception handling functions of the LR2000. The virtual memory system is implemented using a translation lookaside buffer (TLB) and a group of programmable registers.

Memory Management and Exception Handling

Used with Virtual Memory System

Used with Exception Processing
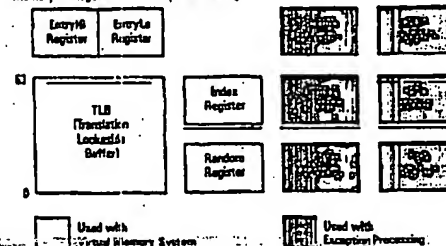
Figure 2. CP0 Registers

Coprocessors

Three types of coprocessor instructions are supported: loads and stores, internal operations; and moves between the coprocessors. The LR2000 coprocessors and the main processor share the same instruction stream. Coprocessor instructions are not explicitly passed to a coprocessor by the LR2000. Instead, coprocessors continuously monitor the data bus, receive instruction/data pairs, and decode valid instructions at the same rate as the main processor.

**LR2000
High Performance
RISC Microprocessor
Preliminary**
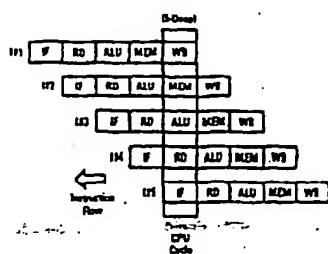
Memory Management System	The LR2000 supports interfaces-to-cache memory and main memory. Often-used operands and instructions are placed into cache memory where the processor can access them quickly. Two direct-mapped caches for instructions (I-cache) and data (D-cache) can range in size from 4 Kbytes to 64 Kbytes. Cache memory access operations take a single cycle to complete. A main memory interface supports reads and writes from/to main (non-cache) memory. The LR2000 has an addressing range of 4 Gbytes (2 Gbytes for the user, 2 Gbytes for the kernel). Since most systems implement physical memory sizes under 4 Gbytes, the LR2000 provides for the logical expansion of memory space by translating	addresses composed in a large virtual address space into available physical memory address. The on-chip translation lookaside buffer provides very fast virtual memory access and is well matched to the requirements of multi-tasking operating systems. The fully-associative TLB contains 64 entries, each of which maps a 4-Kbyte page, with controls for read/write access, cacheability and process identification. The D-cache can be isolated from main memory. The processor also allows swapping of the instruction and data caches. Both operations are used to support cache flushing, diagnostics and trouble shooting.
Pipeline Architecture	The execution of a single LR2000 instruction consists of five primary steps: IF Fetch the instruction (I-cache). RD Read any required operands from CPU registers while decoding the instruction. ALU Perform the required operation of instruction operands. MEM Access memory (D-cache). WB Write results back to the register file. Each of these steps requires on average of one CPU cycle. The LR2000 uses a five-stage pipeline to achieve an instruction execution rate approaching one instruction per CPU cycle. This pipeline operates efficiently because different CPU	resources (address and data bus accesses, ALU operations, register accesses, etc.) are utilized on a non-interfering basis. Even load and store operations execute in a single cycle. Figure 3. Instruction Pipeline
Memory System Hierarchy	The LR2000 supports a high-performance memory hierarchy which centers on the use of external caches. Separate data and instruction caches allow the processor to obtain data and instructions at the CPU cycle rate. These caches are built using commercially available high-speed static RAMS. To en-	sure data consistency, all data written into cache should be written through into main memory. Optional LR2020 write buffers are four-deep, 32-bit write buffers which capture output data from the CPU and ensure its passage on to main memory.
Software and Development Support	The UMIPS operating system is licensable in compiled form from LSI Logic and in source code form from MIPS Computer Systems, Inc. UMIPS is available in both System V.3 and 4.3BSD. UMIPS includes the full complement of UNIX software development utilities such as text editing, source code checking, source code debugging, performance analysis, document formatting, software	project management and compiler generation. Compilers for C, Pascal, FORTRAN, Ada, COBOL and PL-1 are available from LSI Logic or third parties. Board-level products are also available to use as machine code compatible execution vehicles to verify correctness and performance of machine-level instructions. For software and applications development the M/800 and M/1000 systems are available.

MR0105707

LSI LOGIC

LR2000
High Performance
RISC Micro-processor
Preliminary

Pin Descriptions	(Note: an asterisk* indicates an Active-LOW signal)

Data Bus (D31:00)
The 32-bit bidirectional data bus carries all data and instructions between the CPU, caches, main memory and coprocessors.

Data Parity (DataP3:0)
The 4-bit bidirectional data parity bus provides even parity for each of the four bytes of the 32-bit data bus. A data parity error is treated as a cache miss.

Address Low Bus (AdrLo15:00)
The 16-bit AdrLo output carries low-order address bits to the caches and memory subsystem. Only the 14 most significant bits are used to access cache locations. All 16 bits are used for main memory accesses along with 16 bits of the tag bus to form a 32-bit physical memory address. The AdrLo bus is set to high impedance when reset* is asserted or when the processor is brought out of reset in the test state.

Tag Bus (Tag31:12)
The 20-bit tag bus transfers cache tags into the CPU during cache reads. During cache writes, the tag bus carries tag bits into the cache. For main memory accesses, the 16 most significant bits are combined with the AdrLo bus to form a 32-bit physical address.

Tag Valid (TagV)
TagV carries the valid bit between the LR2000 and the caches. During write operation, TagV is HIGH when writing a full 32-bit word to cache and LOW otherwise. During cache reads, TagV is used as one of the criteria in determining whether a cache hit has occurred.

Tag Parity Bus (TagP2:0)
This 3-bit bidirectional bus contains even parity for Tag31–12 and TagV. Tag parity is generated for cache writes and checked during caches reads. A tag parity error is treated as a cache miss.

I-Cache Read (IRd)
D-Cache Read (DRd)
These outputs are asserted during I-cache and D-cache read operations to enable the outputs of the cache RAMs.

I-Cache Write (IWr)
D-Cache Write (DWr)
These outputs are asserted during I-cache and D-cache write operations. These signals are typically used as the write-enable or write-strobe input to the cache RAMs.

I-Cache Latch Clock (ICIk*)
D-Cache Latch Clock (DCIk*)
These outputs are asserted during every cycle. These signals are used to latch addresses into external latches and onto the address bus for the cache RAMs.

Access Type (AccTyp2:0)
AccTyp1 and AccTyp0 indicate the data size for memory accesses and processor-coprocessor transfers as shown below.

Table 2. Access Type Bit Decoding

AccTyp 1 0	Data Size
0 0	Byte (8 bits)
0 1	Half-word (16 bits)
1 0	Three bytes (24 bits)
1 1	Word (32 bits)

AccTyp2 indicates the purpose of an access. During stall cycles, when main memory read is as a result of an I-cache miss, (AccTyp2 is HIGH) or as a result of a D-cache miss (AccTyp2 is LOW).

During run cycles, when the processor data bus will be used during the current cycle, AccTyp2 is LOW, otherwise AccTyp2 is HIGH.

Memory Write (MemWr*)
The MemWr* output is LOW when the processor is performing any write-to-memory. This signal indicates that the tag and address-low buses contain a valid byte address.

Memory Read (MemRd*)
The MemRd* input is LOW when the processor is performing any read-from-memory. This indicates that the tag and address-low buses contain a valid byte address.

Write Busy (WrBusy)
The main memory subsystem places the WrBusy* input LOW to inform the processor that it is not able to accept write data. If the processor needs to perform a write operation while WrBusy* is LOW, the processor stalls until WrBusy* becomes HIGH.

Read Busy (RdBusy)
The main memory subsystem places RdBusy input HIGH to indicate that it is not ready to supply read data requested by the processor. Whenever there is a cache miss, the processor always initiates a read stall while it performs a main memory read. When RdBusy is HIGH it causes the processor to remain in a main memory read stall until it goes LOW.

6

MR0105708

Pin Descriptions (Continued)	

Run*
The Run* input is LOW when the processor is performing a run cycle and is HIGH when the processor is performing stall cycles.

Exception*
The Exception* output is LOW when the processor is responding to an exception and its instruction pipeline has been disrupted. Coprocessors are expected to terminate any instructions in their pipelines.

Coprocessor Busy (CpBusy*)
This input is set LOW by the coprocessor if it needs more time to resolve a data dependency in the instruction stream. When this occurs, the processor initiates a stall which is terminated when CpBusy* goes HIGH.

Coprocessor Condition (CPCond3:0)
The four Coprocessor Condition inputs are generated by up to four coprocessors and used by the LR2000 as condition inputs and are tested during coprocessor branch instructions. The corresponding coprocessor usable bit (Cu3..Cu0) in the status register must be set in order to test one of these condition inputs. Certain software that uses the floating-point coprocessor expects that the CpCond1 input is driven by the LR2010 floating-point coprocessor.

BusError*
This input indicates that a bus error (such as a bus time-out or invalid physical address) has occurred during a RdBusy or WrBusy* stall and causes either a data or instruction bus error exception. The BusError* input is to be used only with synchronous events such as cache miss refills, uncached references and unbuffered writes. A bus error resulting from a buffered write must be signaled using one of the interrupt inputs since the processor is not in a stall and the address that caused the bus error may not still be available to the processor.

Reset*
Reset* is the synchronous initialization input. It must be LOW for a minimum of six cycles to guarantee correct processor initialization, and it must go HIGH with the LR2000 clocks. When Reset* is LOW, the processor initiates a non-maskable exception and subsequently proceeds to reinitialize the system using a predefined bootstrap routine.

Interrupt 0 (Intr0*)
When Reset* is HIGH, the value of Intr0* determines byte ordering or endianness. A HIGH results in a little endian ordering and a LOW results in big endian ordering.

Interrupt 1 (Intr1*)
When Reset* is HIGH, a LOW on Intr1* causes the processor to place all outputs into high impedance to allow external logic to drive signals for board-level testing.

Interrupt 2 (Intr2*)
When Reset* is LOW, the value of Intr2* determines whether caches are presumed present for instructions and data.

Interrupt 3 (Intr3*)
When Reset* is LOW, a LOW on Intr3* causes the processor to place its data and tag outputs into high impedance during write-busy and coprocessor-busy stalls. If Intr3* is HIGH during reset, the data and tag buses are driven during phase 2 of stall cycles. For designs that do not use buses during such stalls, enabling the bus drive prevents the buses from floating for extended periods and avoids overall system design problems.

Interrupt 4 (Intr4*)
When Reset* is LOW, a LOW value of Intr4* causes the processor to insert additional phase delay into its input clock paths. This allows coprocessors to phase lock to the processor and minimize skew.

Interrupt 5 (Intr5*)
Intr5* must be held HIGH during phase 2 while Reset* is HIGH. This will maintain compatibility with future product revisions.

SysOut*, CpSync*
Synchronizing Clock Outputs.

Clk2xSys, Clk2xSmp, Clk2xRd, Clk2xPhi
Four clock inputs. These can be adjusted to obtain optimal positioning of cache interface signals. The relative differences between the clocks are more important than the absolute clock timing. These differences are used to establish the parameters for cache timing.

MR0105709

LR2000
High Performance
RISC Microprocessor
Preliminary

L511B0 0 0 0 0

Operating Parameters | **Absolute Maximum Ratings[1]**

Parameter	Description	Min	Max	Units
VCC	Supply Voltage	-0.5	+7.0	V
VIN	Input Voltage	-0.5[1]	+7.0	V
TST	Storage Temperature	-65	+150	C
TA	Operating Temperature	0	+70	C
CLO	Load Capacitance on Any Pin		100	pF

Operating Range

Range	Ambient Temperature	VCC
Commercial	0°C to 70°C	5V ± 5%

Notes:
1. Operation beyond the limits set forth in this table may impair the useful life of the device.
2. VIN Min. = -3.0 V for pulse width less than 15 ns.
3. Not more than one output should be shorted at a time. Duration of the short should not exceed 30 seconds.

DC Characteristics

Parameter	Description	Test Conditions	12.5 MHz Min	12.5 MHz Max	16.67 MHz Min	16.67 MHz Max	Units
VOH	Output High Voltage	VCC = Min. IOH = -4 mA	3.5		3.5		V
VOL	Output Low Voltage	VCC = Min. IOL = 4 mA		0.4		0.4	V
VIH	Input High Voltage		2.0	VCC + 0.5	2.0	VCC + 0.5	V
VIL	Input Low Voltage		-0.5[1]	0.8	-0.5[1]	0.8	V
VIHS	Input High Voltage		-2.5[1]	VCC + 0.5	3.0[1]	VCC + 0.5	V
VILS	Input Low Voltage		-0.5[1]	0.4	-0.5[1]	0.4	V
CIn	Input Capacitance		10		10		pF
COut	Output Capacitance		10		10		pF
IDD	Operating Current	VCC = Min.		250		300	mA

Notes:
1. VIL Min. = -3.0 V for pulse width less than 15 ns.
2. VIHS and VILS apply to Clk2xSys, Clk2xSmp, Clk2xRd, Clk2xPhi, CpBusy, and Reset*.

MR0105710

LR2000
High Performance
RISC Microprocessor
Preliminary

AC Specifications

Tables 3 through 5 list the preliminary ac electrical specifications for the LR2000. All timings are referenced to 1.5 V. All output timings assume 25 pF of capacitive load. Output timings should be derated where appropriate using the values provided in Table 6.

Table 3. Clock Parameters (Refer to Figure 4)

Parameter	Symbol	Test Conditions	12.5 MHz Min	12.5 MHz Max	16.667 MHz Min	16.667 MHz Max	Units
Input Clock High	TClkHigh	Transition ≤ 5 ns	18		12		ns
Input Clock Low	TClkLow	Transition ≤ 5 ns	16		12		ns
Clock Period	TClkP		40	1000	30	1000	ns
Ch2xSys to Ch2xSmp	TSys-Smp		0	tCyc ÷ 4	0	tCyc ÷ 4	ns
Ch2xSmp to Ch2xRd	TSmp-Rd		0	tCyc ÷ 4	0	tCyc ÷ 4	ns
Ch2xSmp to Ch2xPhi	TSmp		11	tCyc ÷ 4	9	tCyc ÷ 4	ns

The clock parameters apply to all four 2xClocks: Ch2xSys, Ch2xSmp, Ch2xRd, and Ch2xPhi.

Table 4. Run Operation Parameters (Refer to Figures 5-8)

Parameter	Load (pF)	Symbol	12.5 MHz Min	12.5 MHz Max	16.667 MHz Min	16.667 MHz Max	Units	Offset from SysOut*
Data/Tag Valid	25	tDVal	2	3.5	2	3	ns	TSys
Data/Tag Enable		tDEn	−1	−2.5	−1	−2	ns	TSys
Data/Tag Disable		tDDis	0	−1	0	−1	ns	TRd-TSys
Write Delay	25	tWrDly	0	7.5	0	5	ns	TSmp-TSys
Data Setup		tDS	11.5		9		ns	TSmp-TSys
Data Hold		tDH	−4		−4		ns	TSmp-TSys
CpBusy Setup		tCBS	15		13		ns	TSmp-TSys
CpBusy Hold		tCPH	−4		−4		ns	TSmp-TSys
Access Type [1:0]	25	tAcTy	1	10	1	7	ns	TSys
Access Type [2]	25	tAcTy2	1	20	1	17	ns	TSys
Memory Write	25	tMWr	1	10	1	7	ns	TSys
Exception	25	tExc	1	10	1	7	ns	TSys

Table 5. Stall Operation Parameters

Parameter	Load (pF)	Symbol	12.5 MHz Min	12.5 MHz Max	16.667 MHz Min	16.667 MHz Max	Units	Offset from SysOut*
Address Valid	25	tSAVal		38		30	ns	TSys
Access Type	25	tSAcTy		35		27	ns	TSys
Memory Read Initiate	25	tMRd	1	35	1	27	ns	TSys
Memory Read Terminate	25	tMRdT	1	10	1	7	ns	TSys
Run Terminate	25	tStl	5	25	5	17	ns	TSys
Run Initiate	25	tRun	5	15	5	12	ns	TSys
Memory Write	25	tSMwr	5	35	5	27	ns	TSys
Exception Valid	25	tSExc	5	28	5	20	ns	TSys

Table 6. Capacitive Load Derating

Parameter	Symbol	12.5 MHz Min	12.5 MHz Max	16.667 MHz Min	16.667 MHz Max	Units
Load Derate	CLD	1	2.5	1	2	ns/25 pF

AC Specifications
(Continued)
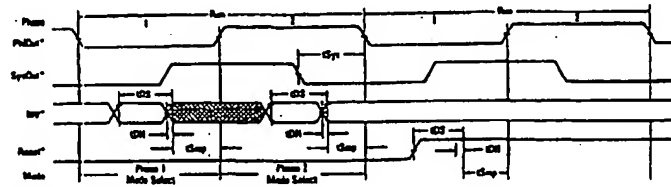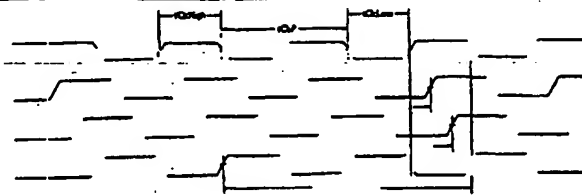
Figure 5. Reset and Mode Select Timing
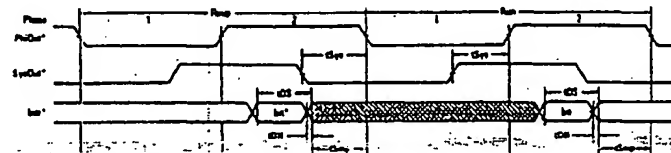
Figure 6. Interrupt Input Timing

MR0105712

**AC Specifications
(Continued)**
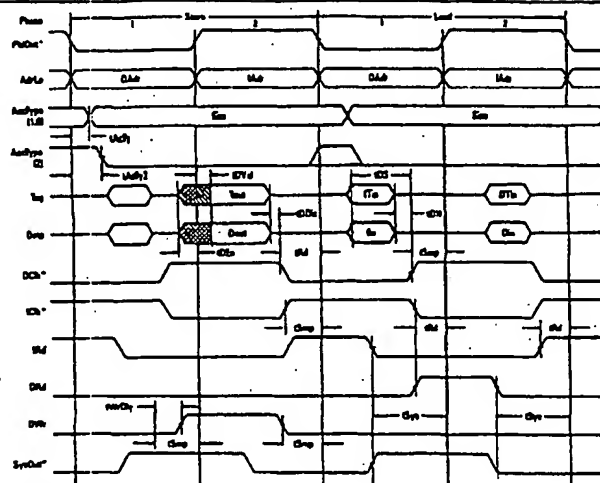
Figure 7. Cache Operation Timing
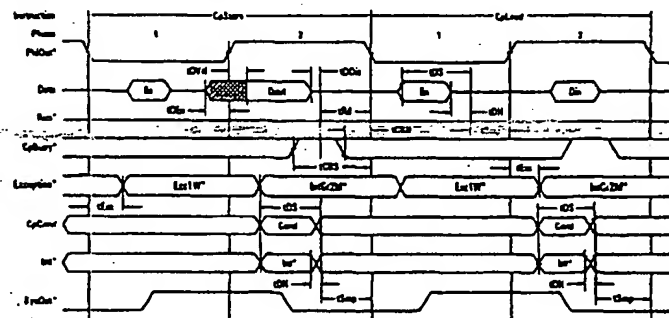
Figure 8. Coprocessor Run Timing

ii

LSI LOGIC

LR2000
High Performance
RISC Microprocessor
Preliminary
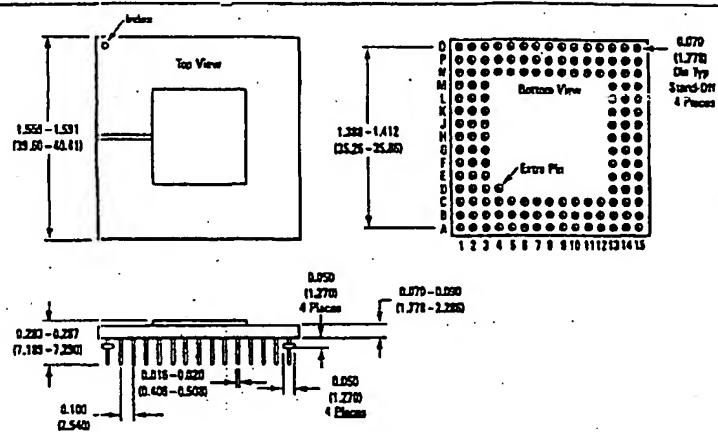
Figure 9. 144 Ceramic Pin Grid Array

12

MR0105714

LR2000
High Performance
RISC Microprocessor
Preliminary

Pin Assignments

Pin Name	Pin Number	Pin Name	Pin Number	Pin Name	Pin Number
Data[0]	E2	Tag[12]	B14	AdrLo[0]	C1
Data[1]	D1	Tag[13]	C13	AdrLo[1]	E3
Data[2]	F3	Tag[14]	D13	AdrLo[2]	D2
Data[3]	G2	Tag[15]	B15	AdrLo[3]	B1
Data[4]	G1	Tag[16]	E13	AdrLo[4]	C2
Data[5]	H2	Tag[17]	D14	AdrLo[5]	C4
Data[6]	H1	Tag[18]	C15	AdrLo[6]	A2
Data[7]	F2	Tag[19]	D15	AdrLo[7]	B3
Data[8]	H3	Tag[20]	E14	AdrLo[8]	C5
Data[9]	J3	Tag[21]	F14	AdrLo[9]	B4
Data[10]	J1	Tag[22]	G14	AdrLo[10]	A3
Data[11]	K2	Tag[23]	F15	AdrLo[11]	A4
Data[12]	L2	Tag[24]	H15	ArdLo[12]	B5
Data[13]	M1	Tag[25]	H14	AdrLo[13]	B7
Data[14]	N1	Tag[26]	J15	AdrLo[14]	A6
Data[15]	K1	Tag[27]	K15	AdrLo[15]	A7
Data[16]	M2	Tag[28]	J13	VCC0	F1
Data[17]	L3	Tag[29]	J14	VCC1	L1
Data[18]	N2	Tag[30]	L15	VCC2	O1
Data[19]	N3	Tag[31]	L14	VCC3	N7
Data[20]	P2	TagP[0]	C14	VCC4	N8
Data[21]	O2	TagP[1]	G15	VCC5	O12
Data[22]	P4	TagP[2]	K14	VCC6	O15
Data[23]	P1	TagV	N15	VCC7	M15
Data[24]	N5	Intr*[0]	C9	VCC8	H13
Data[25]	O3	Intr*[1]	B9	VCC9	E15
Data[26]	P5	Intr*[2]	A11	VCC10	A15
Data[27]	P6	Intr*[3]	B10	VCC11	C8
Data[28]	O5	Intr*[4]	C10	VCC12	A5
Data[29]	O7	Intr*[5]	A12	VCC13	C3
Data[30]	P8	CpCond[0]	A8	VCC14	A1
Data[31]	O4	CpCond[1]	B8	Gnd0	O3
DataP[0]	E1	CpCond[2]	A9	Gnd1	G3
DataP[1]	J2	CpCond[3]	A10	Gnd2	K3
DataP[2]	M3	AccTyp[0]	P15	Gnd3	N4
DataP[3]	N6	AccTyp[1]	M14	Gnd4	O6
Clk2xSys	P9	AccTyp[2]	L13	Gnd5	M9
Clk2xSmp	O10	MemWr*	N12	Gnd6	K10
Clk2xRd	P10	MemRd*	N13	Gnd7	M13
Clk2xPhi	O9	Run*	A14	Gnd8	K13
RdBusy*	C11	IRd*	P12	Gnd9	G13
WrBusy*	A13	IWr*	P13	Gnd10	F13
CpBusy*	B11	ORd*	N11	Gnd11	C12
BusError*	B12	OWr*	O14	Gnd12	C7
Reset*	A14	IClk*	O13	Gnd13	C6
SysOut*	O10	DClk*	P11	Exc*	O8
CpSync*	P14	reserved0	P3	reserved1	P7
reserved2	B2	reserved3	B6	reserved4	B13

Notes:
An asterisk * indicates an Active-LOW signal.
To ensure compatibility with future versions of the LR2000, make no connections to pins labeled reserved.

LR2000
High Performance
RISC Microprocessor
Preliminary

LSI LOGIC

Benchmarks

The following benchmarks illustrate the performance advantage of the LR2000 processor versus other CISC- and RISC-based machines in use today.

These benchmarks are based on industry standard benchmarking programs which are "compute-bound" to measure CPU performance (rather than I/O performance).

The LR2000/12 is the processor used in the M/800 machine, and the LR2000/16 is the processor used in the M/1000 machine. Both machines use the LR2010 Floating-Point Accelerator (FPA).

Dhrystone 1.1 Benchmark
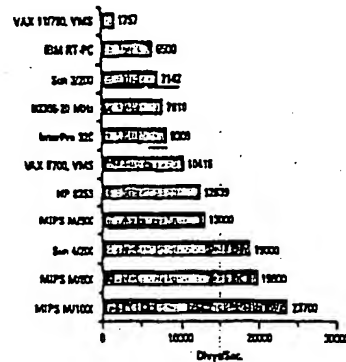
Stanford Integer Benchmark

MR0105716
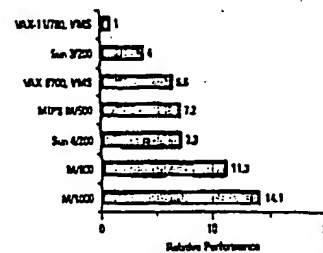
**Benchmarks
(Continued)**

Fortran Elas Linpack Benchmark

MFLOPS

○ Single Precision
▣ Double Precision

Whetstone Benchmark

MWIPS

○ Single Precision
▣ Double Precision
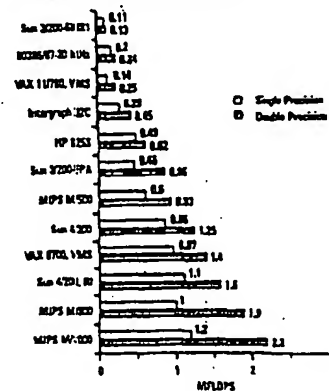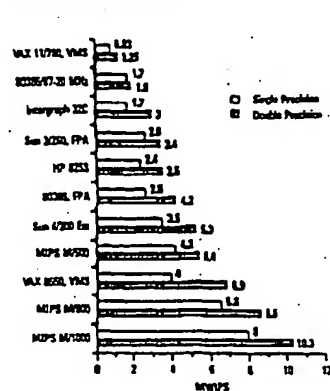
Spice MOSAMP2 Benchmark

Relative Performance
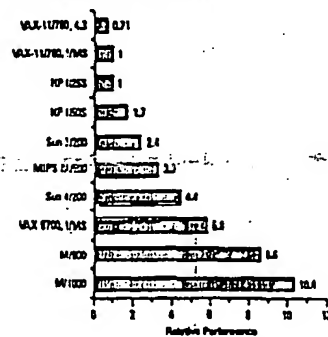
Digital Review Benchmarks
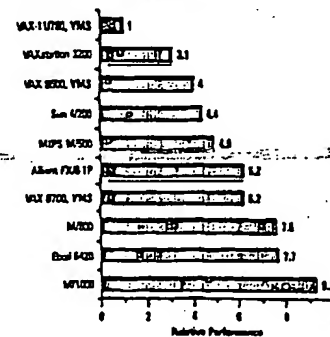
Relative Performance

15

MR0105717

LR2000
High Performance
RISC Microprocessor
Preliminary

LSI LOGIC

Sales Offices
and Design
Resource Centers

LSI Logic Corporation
Headquarters
Milpitas CA
● 408.433.8000

Arizona
602.951.4560

California
San Jose
408.248.5100

Irvine
● 714.553.5600

Sherman Oaks
● 818.906.0333

Colorado
303.758.8800

Florida
Altamonte Springs
407.339.2242

Boca Raton
● 407.395.6200

Georgia
404.448.4858

Illinois
● 312.773.0111

Maryland
● 301.897.5800

Massachusetts
● 617.890.0180 (Design Ctr)
617.890.0181 (Sales Ofc)

Michigan
313.930.6975

Minnesota
● 612.921.8300

New Jersey
● 201.549.4500

New York
914.454.6523

North Carolina
● 919.872.9400

Ohio
614.438.2644

Oregon
503.644.6537

Pennsylvania
215.245.4705

Texas
Austin
512.338.2140

Dallas
● 214.788.2988

Washington
● 206.822.4384

Austria
LSI Logic/Steiner
● 43.222.827474.0

LSI Logic Corporation
of Canada, Inc.
Headquarters
Calgary
● 403.262.9292

Edmonton
● 403.450.4400

Ottawa
● 613.592.1263

Montreal
● 514.694.2417

Toronto
● 416.672.0403

Vancouver
● 604.433.5705

France
LSI Logic S.A.
● 33.1.46212525

Israel
LSI Logic Limited
● 972.3.5403741

Italy
LSI Logic SPA
● 39.39.651575

Japan
LSI Logic K. K.
Tokyo
● 81.3.589.2711

Tsukuba-Shi
● 81.298.52.8371

Osaka
● 81.6.947.5281

LSI Logic Corporation
of Korea Limited
● 82.2.785.1693

Netherlands
LSI Logic/Arechel
● 31.4120.30335

Scotland
LSI Logic Limited
● 44.506.415767

Sweden
LSI Logic Limited
● 46.8.703.4680

Switzerland
LSI Logic/Selzer
● 41.32.515441

United Kingdom
LSI Logic Limited
● 44.344.426544

West Germany
LSI Logic GmbH
Headquarters
Munich
● 49.89.926903.0

Dusseldorf
● 49.211.5961066

Stuttgart
● 49.711.2262151

LSI Logic/EKB
Berlin
● 49.30.311008.0

LSI Logic/Perifiest
Isernhagen
● 49.511.6104.0

LSI Logic/TEP Elektronik
Ingenieurtechnik
Luebeck
● 49.451.893941

● Sales Offices with
Design Resource Centers

MR0105718

863 FH PG 0912

LR2010
Floating-Point
Accelerator
Preliminary

Description

The LR2010 Floating-Point Accelerator (FPA) provides high-speed, floating-point capability for systems based on the LR2000 CPU. The organization of FPA architecture is similar to that of the CPU, allowing high-level language compilers to optimize both integer and floating-point performance. The LR2010, with associated system software, fully conforms to the requirements and recommendations of the ANSI/IEEE Standard 754-1985. The LR2010 connects seamlessly to the CPU. Since both units receive instructions in parallel, floating-point instructions can be initiated at the same single cycle rate as fixed-point instructions.

Features

- Fully compatible to ANSI/IEEE Standard 754-1985 floating-point arithmetic
- Supports single and double precision data formats
- High speed throughput, low latency
- Two speed versions

LR2010LC-12	12.5 MHz
LR2010LC-16	16.7 MHz

- Highly pipelined architecture coupled with optimizing compilers generates high throughput.
- Load/store oriented instruction set initiates floating-point instructions in a single cycle and overlaps execution with additional fixed or floating-point instructions.
- Status/control registers implemented to provide access to all IEEE Standard exception handling capability.
- Sixteen on-chip 64-bit registers individually accessible for flexible operation
- Complete instruction set
  - Single and double precision multiply, divide, add, subtract, negate, absolute value
  - Conversion to/from all supported formats
  - Comparison instructions derived from predicates named in IEEE Standard
- 84-pin ceramic leaded chip carrier
- LR2010 FPA performance floating-point benchmarks
- Linpack

– Single precision	4.8 MFlops
– Double precision	2.2 MFlops

- Whetstone

– Single precision	11.4 MWips
– Double precision	9.1 MWips

- Livermore loops

– Single precision	9.6 × VAX 11/780
– Double precision	12.1 × VAX 11/780

- Spice — 9.7 × VAX 11/780
- 256-Point FFT — 23 × VAX 11/780
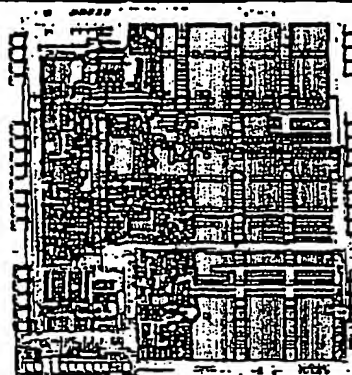
LR2010 FPA Chip Photo

MR0105719

November 1988   Order Number LR2010

863 FH PG 0913

LR2010
Floating-Point
Accelerator
Preliminary
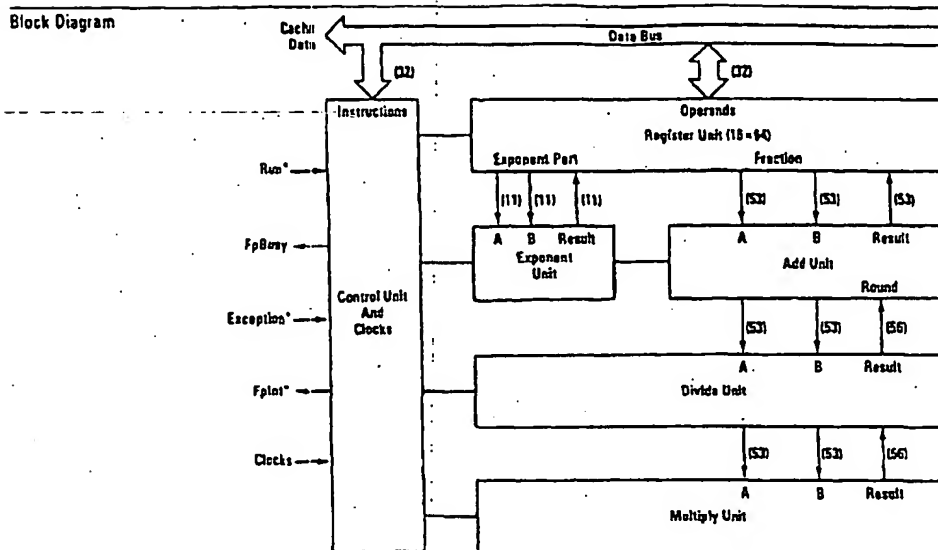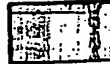
Block Diagram

Note: An asterisk * indicates an Active-LOW Signal.

Figure 1. Functional Block Diagram

Coprocessor
Operation

The LR2010 FPA serves as a seamlessly integrated
coprocessor in floating-point intensive LR2000-
based systems. The FPA continually monitors the
LR2000 instruction stream. If an instruction does
not apply to the coprocessor, it is ignored. If an in-
struction does apply to the coprocessor, the FPA
executes the instruction and transfers results and
necessary exception data synchronously to the
memory. The FPA performs three types of
operations:

- Loads and stores
- Moves
- Two and three-register floating-point operations.

MR0105720

LR2010
Floating-Point
Accelerator
Preliminary

FPA Pipeline Architecture

The execution of a single LR2010 instruction consists of six primary steps:

IF   Instruction Fetch. The main processor calculates the instruction address required to read an instruction from the I-cache. No action is required of the FPA during this pipe stage since the main processor is responsible for address generation.

RD   The instruction is present on the data bus during phase 1 of this pipe stage. The FPA decodes the data and determines whether the instruction will be executed.

ALU   If the decoded instruction applies to the FPA, execution commences during this pipe stage.

MEM   If the instruction is a coprocessor load or store, the FPA captures or presents data during phase 2 of this pipe stage.

WB   The FPA uses this pipe stage to deal with exceptions.

FWB   During this stage the ALU writes results back to the register file. This stage is equivalent to the WB stage in the LR2000 processor.

The LR2010 architecture contains a pipeline similar to the LR2000 processor. The FPA pipeline contains six stages in contrast to the five-stage CPU, providing efficient coordination of exception responses between the FPA and the main processor. Such an architecture operates efficiently because different FPA resources (address and data bus accesses, ALU operations, register accesses, etc.) are utilized on a non-interfering basis. With the use of optimizing compilers to keep the pipeline full, the LR2010 achieves an instruction rate approaching one instruction per second.

IF	RD	ALU	MEM	WB	FWB
I-Cache	RF	OP	D-Cache	Exceptions	FpWB

One Cycle

Figure 2. FPA Instruction Execution Sequence
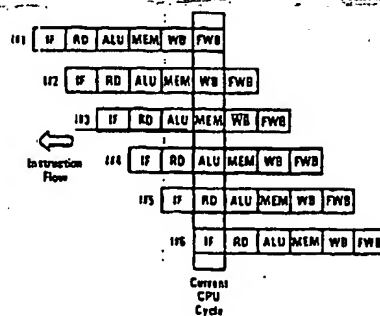
Figure 3. FPA Instruction Pipeline

Programming Model

The LR2010 contains sixteen 64-bit floating-point registers. These are intended to provide a sufficient number of floating-point registers to support allocation of scalar floating-point values and to permit overlapping execution and efficient scheduling of floating-point operations. Each register can hold one value of a single- or double-precision format floating-point number. Extended precision or quad precision floating-point formats can be accommodated by combining adjacent registers.

The coprocessor also contains control and status registers used primarily with diagnostic software, exception handling, state saving and restoring, and control of rounding modes.

The LR2010 FPA provides three types of registers, shown in Figure 4.

Figure 4. FPA Registers

Floating-point general purpose registers (FGR) are directly addressable, physical registers. The FPA provides thirty-two 32-bit FGRs individually accessible via move, load and store operations.

Table 1. Floating-Point General Registers

FGR Number	Usage
0	FPR 0 (Least)
1	FPR 0 (Most)
2	FPR 2 (Least)
3	FPR 2 (Most)
•	•
•	•
•	•
28	FPR 28 (Least)
29	FPR 28 (Most)
30	FPR 30 (Least)
31	FPR 30 (Most)

Floating-point registers (FPR) are logical registers used to store data values for floating-point operations. Each of the FPRs is 64 bits wide and is formed by concatenating two FGRs. The FPRs may hold either single- or double-precision format numbers. Only even-numbered addresses are used to address; odd-numbered register numbers are invalid. During single-precision operations only the even-numbered registers are used. Double-precision operations access general registers in pairs. For example, in a double-precision operation, selecting FPR0 addresses the adjacent floating-point general purpose registers FGR0 and FGR1.

Floating-point control registers (FCR) are used for rounding mode control, exception handling, and state saving. LR2000 coprocessors, in general, can have up to 32 control registers. The FPA implements two: the control/status register (FCR31) and the implementation/revision (FCR0) register.

MR0105722

Programming Model
(Continued)

The control/status register contains control and status data that can be accessed by instructions running in either kernel or user mode. It controls the arithmetic rounding mode, the enabling of exceptions, and exception status. Bit assignments are shown in Figure 5.

The bits in the control/status register can be set or cleared by writing to the register using a move control to coprocessor 1 (ctc1) instruction. The register must only be written to when the FPA is not actively executing floating-point operations. This can be assured by first reading the contents of the register to empty the pipeline. If a floating-point exception occurs as the pipeline empties, the exception is taken and the CFC1 instruction can be re-executed after the exception is serviced.

The FPA control register 0 (FCRO) contains values that define the implementation and revision number of the LR2010 FPA. This information can be used by diagnostic software to determine the coprocessor revision level. Only the low order bytes are defined. Bits 15 through 8 identify the implementation and bits 7 through 0 identify the revision number as shown in Figure 6.

31		16	15		8	7		0
			Imp			Rev		
16				8			8	

Imp Implementation: 0·10·LR2010.
Rev Revision of FPA.
▨ Unused: ignored on writes, zero when read.

Figure 6. Implementation/Revision Register

31	24 23 22	18 17	12 11	7 6	2 1 0
0	C	0	Exceptions E V Z O U I	Trap Enable V Z O U I	Sticky Bits V Z O U I	RM

C	Condition bit. Set/cleared to reflect the result of a compare instruction and drives the FPA CpCond output signal.
Exceptions	These bits are set to indicate any exceptions that occurred during the most recent instructions.
Trap Enable	These bits enable assertion of the Cpint* signal if the corresponding exception bit is set during a floating-point operation.
Sticky Bits	These bits are set if an exception occurs and are reset only by explicitly loading new settings into this register (with a move instruction).
RM	Rounding Mode. These two bits specify which of the four rounding modes is to be used by the FPA.
0	Reserved. Currently ignores writes, undefined when read.

Figure 5. Control/Status Register Bit Assignments

Floating-Point Formats

The LR2010 FPA supports both 32-bit single-precision and 64-bit double-precision IEEE Standard floating-point formats. The 32-bit format has a 24-bit signed magnitude fraction field and an 8-bit exception, as shown in Figure 7.

Figure 7. Single-Precision, Floating-Point Format

The 64-bit format has a 53-bit signed magnitude fraction field and an 11-bit exponent, as shown in Figure 8.
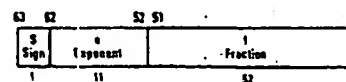
Figure 8. Double-Precision, Floating-Point Format

Floating-point representations in the LR2010 are composed of three fields:

1. A 1-bit sign: s
2. A biased exponent: $e = E + bias$
3. A fraction: $f = .b1b2 \ldots bp\text{-}1$

The range of unbiased exponent E includes every integer between two values EMin and EMax inclusive, and also two other reserved values: EMin − 1 to encode ±0 and denormalized numbers, and EMax + 1 to encode ± ∞ and NaNs (Not-A-Number). For single- and double-precision formats, each representable non-zero value has just one encoding.

The value of a floating-point number is shown in Table 2.

Table 2. Equations for Calculating Values in Floating-Point Format

(1)	If E = EMax + 1 and f ≠ 0, then v is NaN, regardless of s.
(2)	If E = EMax + 1 and f = 0, then v = (−1)^s ∞
(3)	If EMin ≤ E ≤ EMax, then v = (−1)^s 2^E (1.f)
(4)	If E = EMin − 1 and f ≠ 0, then v = (−1)^s 2^{EMin} (0.f)
(5)	If E = EMin − 1 and f = 0, then v = (−1)^s 0

For all floating-point formats, if v is a NaN, the most significant bit of f determines whether the value is a signaling NaN or a quiet NaN. The most significant bit of f will be set for signaling NaN.

The values for the parameters described are shown in Table 3.

Table 3. Floating-Point Format Parameter Values

Parameter	Single	Double
P	24	53
EMax	+127	+1023
EMin	−126	−1022
Exponent Bias	+127	+1023
Exponent Width in Bits	8	11
Integer Bit	Hidden	Hidden
Fraction Width in Bits	23	52
Format Width in Bits	32	64

LR2010
Floating-Point
Accelerator
Preliminary

Number Definitions	The IEEE Standard 754-1985 specifies four varieties of numbers that must be represented: normalized numbers, denormalized numbers, infinity, and zero. The definition of each number type in the LR2010 follows:

Normalized Numbers
Most floating-point calculations are performed on normalized numbers. For single-precision operations, normalized numbers have a biased exponent that ranges from 1 to 254 (−126 to +127 unbiased) and a normalized fraction field, meaning that the leftmost (hidden) bit is one. In decimal notation this allows representation of a range of positive and negative values from approximately 10^{38} to 10^{-38}, with accuracy to seven decimal places.

Denormalized Numbers
Denormalized numbers have a zero exponent and a denormalized (hidden bit = 0) non-zero fraction field.

Infinity
Infinity has an exponent of all ones and a fraction field equal to zero. Both positive and negative infinity are supported.

Zero
Zero has an exponent of zero, a hidden bit equal to zero, and a value of zero in the fraction field. Both +0 and −0 are supported.

Instruction Set Summary	The floating-point instructions supported by the LR2010 are all implemented using the coprocessor unit 1 (COP1) operation instructions of the LR2000 CPU instruction set. The basic operations performed by the CPU are:

- Load/store operations from/to the FPA registers
- Moves between the CPU and the FPA registers
- Computational instructions including floating-point add, subtract, multiply, divide and convert instructions
- Floating-point comparisons

Load, Store and Move Instructions
All movement of data between the LR2010 FPA and memory is accomplished by load word to coprocessor 1 (LWC1) and store word to coprocessor 1 (SWC1) instructions which reference a single 32-bit word of the FPA's general registers. These loads and stores are unformatted; no format conversions are performed and therefore no floating-point exceptions occur due to these operations.

Instruction Set Summary (Continued)

Data may also be directly moved between the FPA and the LR2000 CPU by the move to coprocessor 1 (MTC1) and move from coprocessor 1 (MFC1) instructions. Like the floating-point load and store operations, these operations perform no format conversions and never cause floating-point exceptions. The load and move instructions have a latency of one instruction. Data being loaded from

memory or the CPU into an FPA register is not available to the instruction that immediately follows the load instruction. Data becomes available to the second instruction following the load.

Table 4 summarizes the LR2010 load, store and move instructions.

Table 4. FPA Load, Store and Move Instruction Summary

Instruction	Format and Description
Load Word to FPA (Coprocessor 1)	*LWC1 ft,Offset(Base)* Sign-extend 16-bit *offset* and add to contents of CPU register *base* to form address. Load contents of addressed word into FPA general register *ft*.
Store Word from FPA (Coprocessor 1)	*SWC1 ft,Offset(Base)* Sign-extend 16-bit *offset* and add to contents of CPU register *base* to form address. Store 32-bit contents of FPA general register *ft* at addressed location.
Move Word to FPA (Coprocessor 1)	*MTC1 rt,fs* Move contents of CPU register *rt* into FPA register *fs*.
Move Word from FPA (Coprocessor 1)	*MFC1 rt,fs* Move contents of FPA general register *fs* into CPU register *rt*.
Move Control Word to FPA (Coprocessor 1)	*CTC1 rt,fs* Move contents of CPU register *rt* into FPA control register *fs*.
Move Control Word from FPA (Coprocessor 1)	*CFC1 rt,fs* Move contents of FPA control register *fs* into CPU register *rt*.

Instruction Set Summary (Continued)

Computational Instructions

Computational instructions perform arithmetic operations on floating-point values in registers. There are four categories of floating-point computational instruction:

- 3-operand register-type instructions that perform floating-point addition, subtraction, multiplication and division operations.
- 2-operand register-type instructions that perform floating-point absolute value, move and negate operations

- Convert instructions that perform conversions between the various formats
- Compare instructions that perform comparisons of the contents of two registers and set or clear a condition flag based on the result of the comparison.

Table 5 summarizes the computational instructions. The fmt term appended to the instruction op code is the data format specifier: s specifies single-precision binary floating point, d specifies double-precision binary floating point, and w specifies fixed point. When fmt is single precision or fixed point, the odd register of the destination is undefined.

Table 5. FPA Computational Instruction Summary

Instruction	Format and Description
Floating-Point Add	ADD.fmt fd,fs,ft Interpret contents of FPA registers fs and ft in specified format (fmt) and add arithmetically. Place rounded result in FPA register fd.
Floating-Point Subtract	SUB.fmt fd,fs,ft Interpret contents of FPA registers fs and ft in specified format (fmt) and arithmetically subtract ft from fs. Place result in FPA register fd.
Floating-Point Multiply	MUL.fmt fd,fs,ft Interpret contents of FPA registers fs and ft in specified format (fmt) and arithmetically multiply ft and fs. Place result in FPA register fd.
Floating-Point Divide	DIV.fmt fd,fs,ft Interpret contents of FPA registers fs and ft in specified format (fmt) and arithmetically divide fs by ft. Place rounded result in register fd.
Floating-Point Absolute Value	ABS.fmt fd,fs Interpret contents of FPA register fs in specified format (fmt) and take arithmetic absolute value. Place result in FPA register fd.
Floating-Point Move	MOV.fmt fd,fs Interpret contents of FPA register fs in specified format (fmt) and copy into FPA register fd.
Floating-Point Negate	NEG.fmt fd,fs Interpret contents of FPA register fs in specified format (fmt) and take arithmetic negation. Place result in FPA register fd.
Floating-Point Convert to Single FP Format	CVT.S.fmt fd,fs Interpret contents of FPA register fs in specified format (fmt) and arithmetically convert to the single binary floating-point format. Place rounded result in FPA register fd.
Floating-Point Convert to Double FP Format	CVT.D.fmt fd,fs Interpret contents of FPA register fs in specified format (fmt) and arithmetically convert to the double binary floating-point format. Place rounded result in FPA register fd.
Floating-Point Convert to Single Fixed-Point Format	CVT.W.fmt fd,fs Interpret contents of FPA register fs in specified format (fmt) and arithmetically convert to the single fixed-point format. Place result in FPA register fd.
Floating-Point Compare	C.cond.fmt fs,ft Interpret contents of FPA registers fs and ft in specified format (fmt) and arithmetically compare. The result is determined by the comparison and the specified condition (cond). After a one instruction delay, the condition is available for testing by the CPU with the branch on floating-point coprocessor condition (BC1T, BC1F) instructions.

Instruction Set Summary (Continued)

Floating-Point Relational Operations

The floating-point compare instructions (C.fmt.cond) interpret the contents of two FPA registers (fs, ft) in the specified format (fmt) and arithmetically compare them. The result is based on the comparison and the conditions (cond) specified in the instruction. Table 6 lists the conditions that can be specified for the compare instruction and Table 7 summarizes the floating-point relational operations that may be performed.

Table 7 is derived from a similar table in the IEEE Standard and describes 26 predicates named in the standard. The table also includes six additional predicates to round out the set of possible predicates based on a condition tested by a comparison. Four mutually exclusive relations are possible:

less than, greater than, equal, and unordered. Note that invalid operations occur only when the comparisons include the less-than and greater-than characters but not the unordered character in the ad hoc form of the predicate.

Branch on FPA Condition Instructions

Table 8 summarizes the two branch on FPA (coprocessor unit 1) condition instructions that can be used to test the result of the FPA compare instructions. The term delay slot, described in the table, refers to the instruction immediately following the branch instruction.

Table 6. Relational Mnemonic Definitions

Mnemonic	Definition	Mnemonic	Definition
F	False	T	True
UN	Unordered	OR	Ordered
EQ	Equal	NEQ	Not Equal
UEQ	Unordered or Equal	OLG	Ordered or Less Than or Greater Than
OLT	Ordered Less Than	UGE	Unordered or Greater Than or Equal
ULT	Unordered or Less Than	OGE	Ordered Greater Than
OLE	Ordered Less Than or Equal	UGT	Unordered or Greater Than
ULE	Unordered or Less Than or Equal	OGT	Ordered Greater Than
SF	Signaling False	ST	Signaling True
NGLE	Not Greater Than or Less Than or Equal	GLE	Greater Than, or Less Than or Equal
SEQ	Signaling Equal	SNE	Signaling Not Equal
NGL	Not Greater Than or Less Than	GL	Greater Than or Less Than
LT	Less Than	NLT	Not Less Than
NGE	Not Greater Than or Equal	GE	Greater Than or Equal
LE	Less Than or Equal	NLE	Not Less Than or Equal
NGT	Not Greater Than	GT	Greater Than

Instruction Set
Summary
(Continued)

Table 7. Floating-Point Relational Operators

Condition Mnemonic	Predicates		Relations				Invalid Operation Exception if Unordered
	Ad Hoc	FORTRAN	Greater Than	Less Than	Equal	Unordered	
F	false		F	F	F	F	no
UN	?		F	F	F	T	no
EQ	=	.EQ.	F	F	T	F	no
UEQ	?=	.UE.	F	F	T	T	no
OLT	NOT(?>=)	.NOT..UG.	F	T	F	F	no
ULT	?<	.UL.	F	T	F	T	no
OLE	NOT(?>)	.NOT..UG.	F	T	T	F	no
ULE	?<=	.ULE.	F	T	T	T	no
OGT	NOT(?<=)	.NOT..ULE	T	F	F	F	no
UGT	?>	.UGT.	T	F	F	T	no
OGE	NOT(?<)	.NOT..UL.	T	F	T	F	no
UGE	?>=	.UGE.	T	F	T	T	no
OLG	NOT(?=)		T	T	F	F	no
NEQ	NOT(=)	.NE.	T	T	F	T	no
OR	NOT(?)		T	T	T	F	no
T	true		T	T	T	T	no
SF			F	F	F	F	yes
NGLE	NOT(<=>)	.NOT..LEG.	F	F	F	T	yes
SEQ			F	F	T	F	yes
NGL	NOT(<>)	.NOT..LG.	F	F	T	T	yes
LT	<	.LT.	F	T	F	F	yes
NGE	NOT(>=)	.NOT..GE.	F	T	F	T	yes
LE	<=	.LE.	F	T	T	F	yes
NGT	NOT(>)	.NOT..GT.	F	T	T	T	yes
GT	>	.GT.	T	F	F	F	yes
NLE	NOT(<=)	.NOT..LE.	T	F	F	T	yes
GE	>=	.GE.	T	F	T	F	yes
NLT	NOT(<)	.NOT..LT.	T	F	T	T	yes
GL	<>	.LG.	T	T	F	F	yes
SNE			T	T	F	T	yes
GLE	<=>	.LEG.	T	T	T	F	yes
ST			T	T	T	T	yes

Table 8. Branch on FPA Condition Instructions

Instruction	Format and Description
Branch on FPA True	BCIT Compute a branch target address by adding address of instruction in the delay slot and the 16-bit *offset* (shifted left two bits and sign-extended to 32 bits). Branch to the target address (with a delay of one instruction) if the FPA's CpCond signal is true.
Branch on FPA False	BCIF Compute a branch target address by adding address of instruction in the delay slot and the 16-bit *offset* (shifted left two bits and sign-extended to 32 bits). Branch to the target address (with a delay of one instruction) if the FPA's CpCond signal is false.

II

MR0105729

LR2010
Floating-Point
Accelerator
Preliminary

Instruction Execution Times	Unlike the LR2000 which executes nearly all its instructions in a single cycle, the time to execute an FPA instruction ranges from 1 cycle to 19 cycles. Figure 9 illustrates the number of cycles required to execute each of the FPA instructions. The cycles of an instruction's execution time that are darkly shaded require exclusive access to an FPA resource that precludes concurrent use by another	instruction. With the exception of loads and stores, other FPA instructions cannot be overlapped during these cycles. Those instruction cycles that are lightly shaded place minimal demands on FPA resources and may be overlapped (with some exceptions) to obtain simultaneous execution without stalling the pipeline.
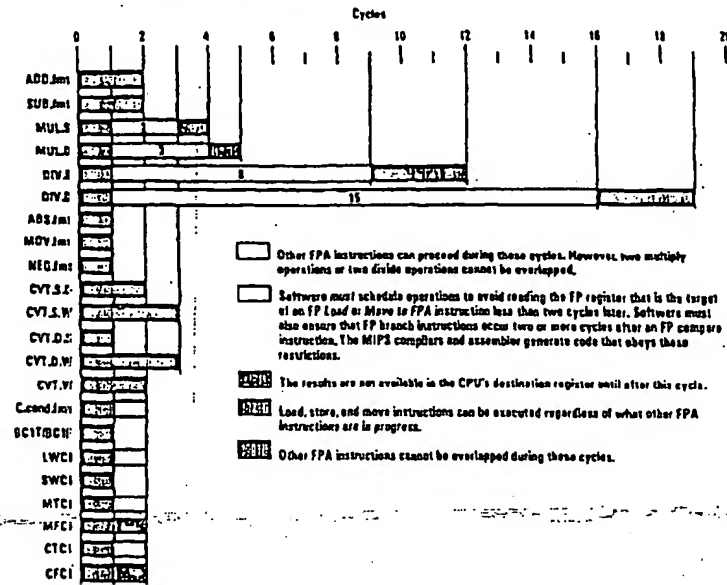
Figure 9. FPA Instruction Execution Times

MR0105730

863 FH PG 0924

LR2010
Floating-Point
Accelerator
Preliminary

Overlapping FPA Instructions

Figure 10 illustrates the overlapping of several FPA (and non-FPA) instructions. In this example, the first instruction requires 12 total cycles for execution but only the first cycle and the last three cycles inhibit simultaneous execution of other instructions. Similarly, the second instruction (MUL.S) has two cycles in the middle of its total of four required cycles that can be used to advance the execution of the third and fourth instructions.

Although processing of a single instruction consists of six pipe stages, the FPA does not require that the instruction actually be completed in six cycles to avoid stalling the pipeline. If a subsequent instruction does not require the resources being used by a preceding instruction and has no data dependencies on uncompleted instructions, then execution continues.
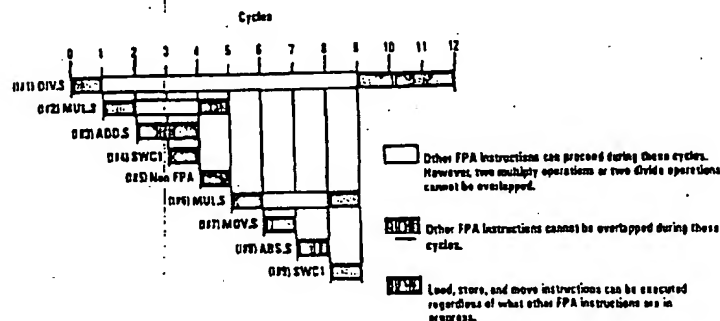
Figure 10. Overlapping FPA Instructions

Floating-Point Exceptions

Floating-point exceptions occur when the FPA cannot handle the results of a floating-point operation in a normal way. The FPA responds by either generating an interrupt or setting a status flag. The control status register previously described contains a trap enable bit for each exception type that determines whether an exception will cause the FPA to initiate a trap or set a status flag. If a trap is taken, the FPA remains in the state found at the beginning of the operation and a software handling routine is executed. If no trap is taken, an appropriate value is written into the FPA destination register and execution continues.

The FPA supports the five IEEE exceptions — inexact (I), overflow (O), underflow (U), divide by zero (Z), and invalid (V) — with exception bits, trap enables and sticky bits. The LR2010 FPA adds a sixth exception type, unimplemented operation (E), to be used in those cases where a software implementation must be employed to conform to the MIPS floating-point architecture. The unimplemented operation exception has no trap enable or sticky bit. Whenever this exception occurs, an unimplemented exception trap is taken (if the FP interrupt input to the LR2000 is enabled).

Figure 11 shows the control/status register associated with the five IEEE exceptions (V,Z,O,I,U). When an exception occurs, the corresponding exception and sticky bits are set. If the corresponding trap enable bit is set, the FPA generates an interrupt to the LR2000 processor and subsequent exception processing allows a trap to be taken.
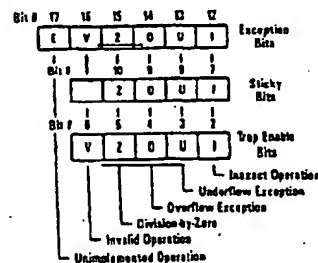
Figure 11. Control/Status Register Exception/Sticky/Trap Enable Bits

MR0105731

Floating-Point Exceptions (Continued)

Exception Trap Processing

When a floating-point exception trap is taken, the LR2000s cause register indicates that an external interrupt is the cause of the exception and the LR2000s EPC (exception program counter) contains the address of the instruction that caused the exception trap.

For each IEEE Standard exception, a sticky-bit status flag is provided that is set on the occurrence of the corresponding condition with no corresponding exception trap signaled. The sticky bits may be reset by writing a new value into the control/status register and may be saved and restored by software.

When no exception trap is signaled, a default action is taken by the FPA which provides a substitute value for the original exceptional result of the floating-point operation. The default action depends on the type of exception and, in the case of overflow, the current rounding mode. Table 10 lists the default action taken by the FPA for each of the IEEE exceptions.

The FPA internally detects eight different conditions that can cause exceptions. When the FPA encounters one of these situations it will cause either an IEEE exception or an unimplemented operation (E) exception. Table 9 lists the exception-causing situations.

The following sections describe the conditions that cause the FPA to generate each of its six exceptions and details the FPAs response to each of these situations.

Table 9. FPA Exception Situations

FPA Internal Result	IEEE Standard	Trap Enabled	Trap Disabled	Note
Inexact Result	I	I	I	Loss of accuracy
Exponent Overflow	O I*	O I	O I	Normalized exponent > EMax
Divide by Zero	Z	Z	Z	Zero is (exponent = EMin - 1, mantissa = 0)
Overflow on Convert	V	V	E	Source out of integer range
Signaling NaN Source	V	V	E	Quiet NaN source produces quiet NaN result
Invalid Operation	V	V	E	0/0 etc.
Exponent Underflow	U	E	E	Normalized exponent < EMin
Denormalized Source	None	E	E	Exponent = EMin - 1 and mantissa < > 0

*Standard specifies inexact exception on overflow only if overflow trap is disabled.

Table 10. FPA Exception Default Actions

	Exception	Rounding Mode	Default Action (No Exception Trap Signaled)
V	Invalid Operation	—	Supply a quiet NaN.
Z	Division by Zero	—	Supply a properly signed ∞.
O	Overflow	RN	Modify overflow values to ∞ with the sign of the intermediate result.
		RZ	Modify overflow values to the format's largest finite number with the sign of the intermediate result.
		RP	Modify negative overflows to the format's most negative finite number. Modify positive overflows to + ∞.
		RM	Modify positive overflows to the format's largest finite number. Modify negative overflows to - ∞.
U	Underflow	—	Generate an unimplemented exception.
I	Inexact	—	Supply a rounded result.

MR0105732

**Floating-Point
Exceptions
(Continued)**

Inexact Exception (I)

The FPA generates this exception if the rounded result of an operation is not exact or if it overflows.

The FPA usually examines the operands of floating-point operations before execution actually begins to determine (based on the exponent values of the operands) if the operation can *possibly* cause an exception. If there is a possibility of an instruction causing an exception trap, then the FPA uses the coprocessor stall mechanism previously described. It is impossible, however, for the FPA to predetermine if an instruction will produce an inexact result. Therefore, if inexact exception traps are enabled, the FPA uses the coprocessor stall mechanism to execute all floating-point operations that require more than one cycle. Since this mode of execution can impact performance, inexact exception traps should be enabled only when necessary.

Trap Enabled Results: If inexact exception traps are enabled, the result register is not modified and the source registers are preserved.

Trap Disabled Results: The rounded or overflowed result is delivered to the destination register if no other software trap occurs.

Underflow Exception (U)

The FPA never generates an underflow exception and never sets the *U* bit in either the *exceptions* field or *sticky* field of the control/status register. If the FPA detects a condition that could be either an underflow or a loss of accuracy, it generates an unimplemented exception.

Overflow Exception (O)

The overflow exception is signaled when what would have been the magnitude of the rounded floating-point result, were the exponent range unbounded, is larger than the destination format's largest finite number. (This exception also sets the inexact exception and sticky bits.)

Trap Enabled Results: The result register is not modified, and the source registers are preserved.

Trap Disabled Results: The result, when no trap occurs, is determined by the rounding mode and the sign of the intermediate result (as listed in Table 10).

Division-by-Zero Exception (Z)

The division-by-zero exception is signaled on a divide operation if the divisor is zero and the dividend is a finite non-zero number.

Trap Enabled Results: The result register is not modified, and the source registers are preserved.

Trap Disabled Results: The result, when no trap occurs, is a correctly signed infinity.

Invalid Operation Exception (V)

The invalid operation exception is signaled if one or both of the operands are invalid for an implemented operation. The invalid operations are:

1. Addition or subtraction: magnitude subtraction of infinities, such as: $(+\infty) - (+\infty)$
2. Multiplication: $0 \times \infty$, with any signs
3. Division: $0 \div 0$, or $\infty \div \infty$, with any signs
4. Conversion of a floating-point number to a fixed-point format when an overflow, or operand value of infinity or NaN, precludes a faithful representation in that format
5. Comparison of predicates involving $<$ or $>$ without ?, when the operands are "unordered"
6. Any arithmetic operation on a signaling NaN. Note that a move (MOV) operation is not considered to be an arithmetic operation, but that ABS and NEG are considered to be arithmetic operations and will cause this exception if one or both operands is a signaling NaN.

Software may simulate this exception for other operations that are invalid for the given source operands. Examples of these operations include IEEE-specified functions implemented in software, such as remainder: x REM y, where y is zero or x is infinite; conversion of a floating-point number to a decimal format whose value causes an overflow or is infinity or NaN; and transcendental functions, such as $\ln(-5)$ or $\cos^{-1}(3)$.

Trap Enabled Results: The original operand values are undisturbed.

Trap Disabled Results: The FPA always signals an unimplemented exception because it does not create the NaN that the IEEE Standard specifies should be returned under these circumstances.

MR0105733

863 FH PG 0927

LR2010
Floating-Point
Accelerator
Preliminary

Floating-Point Exceptions (Continued)	**Unimplemented Operation Exception (E)** The FPA generates this exception when it attempts to execute an instruction with an OpCode (bits 31–26) or format code (bits 24–21) which has been reserved for future use. This exception is not maskable: the trap is always enabled. When an unimplemented operation is signaled, an interrupt is sent to the LR2000 processor so that the operation can be emulated in software. When the operation is emulated in software, any of the IEEE exceptions may arise; these exceptions must, in turn, be simulated. This exception is also generated when any of the following exceptions are detected by the FPA:	■ Extended and quad precision ■ Square root ■ Denormalized operand ■ Not-a-number (NaN) operand ■ Invalid operation with trap disabled ■ Denormalized result ■ Underflow Trap Enabled Results: The original operand values are undisturbed. Trap Disabled Results: This trap cannot be disabled.
Saving and Restoring State	Thirty-two coprocessor load or store instructions will save or restore the FPAs floating-point register state in memory. The contents of the control/status register can be saved using the "move to/from coprocessor control register" instructions (CTC1/CFC1). Normally, the control/status register contents are saved first and restored last. If the control/status register is read when the coprocessor is executing one or more floating-point instructions, the instructions in progress (in the pipeline) are completed before the contents of the register are moved to the main processor. If an exception occurs during one of the in-progress instructions, that exception is written into the control/status register exceptions' field.	Note that the exceptions field of the control/status register holds the results of only one instruction: the FPA examines source operands before an operation is initiated to determine if the instruction can possibly cause an exception. If an exception is possible, the FPA executes the instruction in "stall" mode to ensure that no more than one instruction at a time is executed that might cause an exception. All of the bits in the exceptions field can be cleared by writing a zero value to this field. This permits restarting of normal processing after the control/status register state is restored.

16

LR2010
Floating-Point
Accelerator
Preliminary

Pin Descriptions

(Note: an asterisk * indicates an Active-LOW signal)

Data (31:0)
(I/O) A multiplexed 32-bit bus used for instruction and data transfers on phase 1 and phase 2, respectively.

Data P(3:0)
(O) A 4-bit bus containing even parity over the data bus. Parity is generated by the FPC on stores.

Run*
(I) Input to the FPC which indicates whether the processor-coprocessor system is in the run or stall state.

Exception*
(I) Input to the FPC which indicates exception related status information.

FpBusy
(O) Signal to the CPU indicating a request for a co-processor busy stall.

FpCond
(O) Signal to the CPU indicating the result of the last comparison operation.

FpInt*
(O) Signal to the CPU indicating that a floating-point exception has occurred for the current FPC instruction.

Reset*
(I) Synchronous initialization input used to distinguish the processor-FPC synchronization period from the execution period. Reset* must be synchronized by the leading edge of SysOut from the CPU.

PLLOn*
(I) Input which during the reset period determines whether the phase lock mechanism is enabled and during the execution period determines the output timing model.

FpPresent*
(O) Output which is pulled to ground through an impedance of approximately 0.5K Ω. By providing an external pullup on this line, an indication of the presence or absence of the FPC can be obtained.

Clk2×Sys
(I) A double-frequency clock input used for generating FpSysOut*.

Clk2×Smp
(I) A double-frequency clock input used to determine the sample point for data coming into the FPC.

Clk2×Rd
(I) A double-frequency clock input used to determine the disable point for the data drivers.

Clk2×Phi
(I) A double-frequency clock input used to determine the position of the internal phases; phase 1 and phase 2.

FpSysOut*
(O) Synchronization clock from the FPC.

FpSysIn*
(I) Input used to receive the synchronization clock from the FPC.

FpSync*
(I) Input used to receive the synchronization clock from the CPU.

17

MR0105735

LR2010
Floating-Point
Accelerator
Preliminary

Pin Assignments

Table 11. FPC Pinout 84-Pin Quad J-Lead CerPak

Pin Name	Pin Number	Pin Name	Pin Number
Data(0)	33	FpSync*	23
Data(1)	34	Reset*	22
Data(2)	35	PWDn*	28
Data(3)	36	Run*	66
Data(4)	39	Exception*	67
Data(5)	40	FpInt*	68
Data(6)	41	FpBusy	69
Data(7)	42	FpCond	70
Data(8)	44	VCC0	7
Data(9)	45	VCC1	15
Data(10)	46	VCC2	24
Data(11)	47	VCC3	26
Data(12)	50	VCC4	29
Data(13)	51	VCC5	31
Data(14)	52	VCC6	38
Data(15)	53	VCC7	49
Data(16)	76	VCC8	55
Data(17)	77	VCC9	57
Data(18)	78	VCC10	61
Data(19)	79	VCC11	63
Data(20)	82	VCC12	72
Data(21)	83	VCC13	75
Data(22)	84	VCC14	81
Data(23)	1	Gnd0	6
Data(24)	3	Gnd1	16
Data(25)	4	Gnd2	25
Data(26)	5	Gnd3	27
Data(27)	8	Gnd4	30
Data(28)	9	Gnd5	32
Data(29)	10	Gnd6	37
Data(30)	11	Gnd7	48
Data(31)	14	Gnd8	54
DataP(0)	43	Gnd9	56
DataP(1)	73	Gnd10	60
DataP(2)	2	Gnd11	62
DataP(3)	17	Gnd12	71
Ch2:Sys	19	Gnd13	74
Ch2:Smp	20	Gnd14	80
Ch2:Ad	12	Resrv0	58
Ch2:Phi	21	Resrv1	64
FpSysIn*	13	Resrv2	65
FpSysOut*	18	FpPresent*	59

Note: An asterisk * indicates an Active-LOW signal

Operating Parameters

Absolute Maximum Ratings[1]

Parameter	Description	Min	Max	Units
VCC	Supply Voltage	-0.5	+7.0	V
VIN	Input Voltage	-0.5[2]	+7.0	V
TST	Storage Temperature	-65	+150	C
TA	Operating Temperature	0	+70	C
CLO	Load Capacitance on Any Pin		100	pF

Notes:
1. Operation beyond the limits set forth in this table may impair the useful life of the device.
2. VIN Min. = –3.0 V for pulse width less than 15 ns.
3. Not more than one output should be shorted at a time. Duration of the short should not exceed 30 seconds.

Operating Range

Range	Ambient Temperature	VCC
Commercial	0°C to 70°C	5V ± 5%

18

MR0105736

863 FH PG 0930

LR2010
Floating-Point
Accelerator
Preliminary

OC Characteristics

Parameter	Description	Test Conditions	12.5 MHz		16.67 MHz		Units
			Min	Max	Min	Max	
VOH	Output High Voltage	VCC = Min. IOH = -4 mA	3.5		3.5		V
VOL	Output Low Voltage	VCC = Min. IOL = 4 mA		0.4		0.4	V
VIH	Input High Voltage		2.0	VCC + 0.5	2.0	VCC + 0.5	V
VIL	Input Low Voltage		-0.5[1]	0.8	-0.5[1]	0.8	V
VIHS	Input High Voltage		-2.5[2]	VCC + 0.5	3.0[2]	VCC + 0.5	V
VILS	Input Low Voltage		-0.5[1]	0.4	-0.5[1]	0.4	V
VIHC	Input High Voltage		4.0[3]	VCC + 0.5	4.0[3]	VCC + 0.5	V
VILC	Input Low Voltage		-0.5[1]	0.4	-0.5[1]	0.4	V
Cin	Input Capacitance		10		10		pF
COut	Output Capacitance		10		10		pF
ICC	Operating Current			500		550	mA

Notes:
1. VIL Min. = -3.0 V for pulse width less than 15 ns.
2. VIHS and VILS apply to Ch2xSys, Ch2xSmp, Ch2xRd, Ch2xPh.
3. VIHC and VILC apply to Run* and Exception*

Packaging   84-Pin Quad Type "J" Package
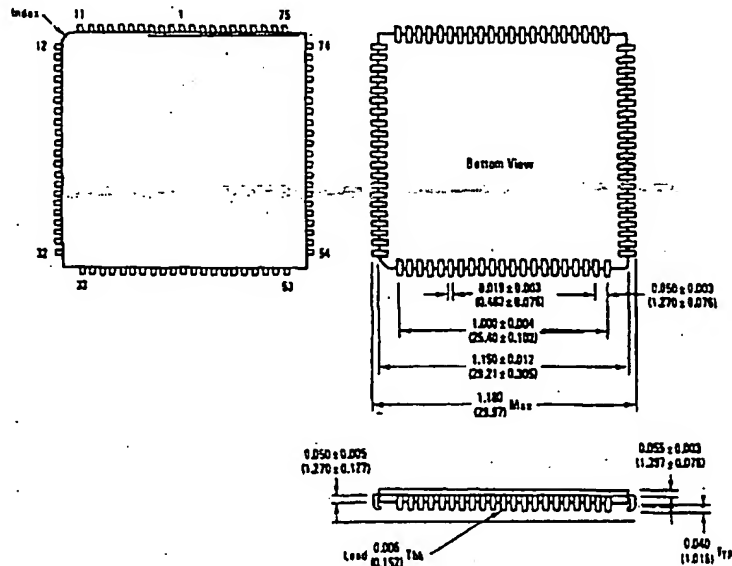
MR0105737

LR2010
Floating-Point
Accelerator
Preliminary

Sales Offices
and Design
Resource Centers

LSI Logic Corporation
Headquarters
Milpitas CA
• 408.433.8000

Arizona
602.951.4560

California
San Jose
408.248.5100

Irvine
• 714.553.5600

Sherman Oaks
• 818.906.0333

Colorado
303.756.8800

Florida
Altamonte Springs
407.339.2742

Boca Raton
• 407.395.6200

Georgia
404.448.4898

Illinois
• 312.773.0111

Maryland
301.897.5800

Massachusetts
• 617.890.0180 (Design Ctr)
617.890.0181 (Sales Ofc)

Michigan
313.930.5975

Minnesota
• 612.921.8300

New Jersey
• 201.549.4500

New York
914.226.1870

North Carolina
• 919.872.8400

Ohio
614.438.2644

Oregon
503.644.5697

Pennsylvania
215.638.3010

Texas
Austin
512.338.2140

Dallas
• 214.788.2955

Washington
• 206.822.4384

Austria
LSI Logic/Steiner
• 43.222.827474.0

LSI Logic Corporation
of Canada, Inc.
Headquarters
Calgary
• 403.262.9292

Edmonton
• 403.450.4400

Ottawa
• 613.592.1263

Montreal
• 514.694.2417

Toronto
• 416.672.0403

Vancouver
• 604.433.5705

France
LSI Logic S.A.
• 33.1.46212525

Israel
LSI Logic Limited
• 972.3.5403741

Italy
LSI Logic SPA
• 39.39.651575

Japan
LSI Logic K. K.
Tokyo
• 81.3.589.2711

Tsukuba-Shi
• 81.298.52.8371

Osaka
• 81.6.947.5281

LSI Logic Corporation
of Korea Limited
• 82.2.785.1693

Netherlands
LSI Logic/Arcobel
• 31.4120.30335

Scotland
LSI Logic Limited
• 44.506.416767

Sweden
LSI Logic Limited
• 46.8.703.4680

Switzerland
LSI Logic/Sulzer
• 41.32.515441

United Kingdom
LSI Logic Limited
• 44.344.426544

West Germany
LSI Logic GmbH
Headquarters
Munich
• 49.89.926903.0

Dusseldorf
• 49.211.5961066

Stuttgart
• 49.711.2262151

LSI Logic/EXB
Berlin
• 49.30.311008.0

LSI Logic/Parizzi
hamshagen
• 49.511.6104.0

LSI Logic/TEP Elektronik
Ingenieurtechnik
Lubeck
• 49.451.893341

MR0105738

863 FH PG 0932

Integrated Device Technology Inc

HIGH-SPEED CMOS DATA BOOK

MR0105692

$\int dt$

$E_{r.}$ $80+$

Integrated
Device
Technology

High Performance CMOS

DATA BOOK
1988

DATA CONVERSION

FAST LOGIC

MICROPROCESSOR

FAST LOGIC

DATA CONVERSION

MR0106691

9.95

863 FH PG 0934

MR0105693

REDUCED INSTRUCTION SET COMPUTER (RISC) PROCESSORS

RISC/CPU PROCESSOR INFORMATION
IDT79R2000

FEATURES:

DESCRIPTION:

R2000 Processor

CPO
(System Control Coprocessor)

CPU

Control

Master Pipeline / Bus Control

Local Control Logic

Exception / Control Registers

Memory Management Unit Registers

Translation Lookaside Buffer (64 entries)

General Registers (32x32)

ALU

Shifter

Multiplier/Divider

Address Adder

PC Increment/Mux

Virtual Page Number/Virtual Address

Data(32+4)

ADDRESS(18)

TAG(20+4)

MILITARY AND COMMERCIAL TEMPERATURE RANGES

TABLE OF CONTENTS

⑨

	DESCRIPTION			DESCRIPTION

Table 1. IDT R3000 Instruction Summary

IDT / R3000 CPU Registers

General Purpose Registers

31   0
r0
r1
r2
.....
r29
r30
r31

Multiply / Divide Registers

HI
LO

Program Counter

PC

Figure 1. IDT R3000 CPU Registers

Instruction Set Overview

I-Type (Immediate)

op	rs	rt	immediate

J-Type (Jump)

op	target

R-Type (Register)

op	rs	rt	rd	re	funct

Figure 2. IDT R3000 Instruction Formats

Kernel Mode Virtual Address Space

0x7fffffff	
	2 GB Mapped
0xe0000000	0.5 GB Unmapped
0xc0000000	0.5 GB Unmapped
0x80000000	
0x7fffffff	2 GB Mapped
0	

User Mode Virtual Address Space

0x7fffffff	2 GB Mapped
0	

Figure 5. IDT79R4000 Virtual Addressing

IDT79R4000 Pipeline Architecture

Instruction Execution

F	RD	ALU	MEM	WB
I-CACHE	RF	OP	D-CACHE	WB

One cycle

Figure 6. Instruction Execution Sequence

CP0 Registers

Memory Management & Exception Handling

Figure 4. The CP0 Registers

System Control Coprocessor (CP0) Registers

Memory Management System

The TLB (Translation Lookaside Buffer)

IDT79R4000 Operating Mode

Table 1. System Control Coprocessor (CP0) Registers

PIN CONFIGURATION

RISC FLOATING POINT
ACCELERATOR (FPA)

ADVANCE INFORMATION
IDT79R3010

FEATURES:

- Provides support of Single and Double-Precision Operations
  - Floating-Point Add
  - Floating-Point Subtract
  - Floating-Point Multiply
  - Floating-Point Divide
  - Floating-Point Conversions
  - Floating-Point Comparisons
- Fast Speed: 70-85 mini cycle, store and move
- Peak Speeds: 10-16 MFLOPS Single or Double-Precision
- Cycle Time: 40-50ns (On-Chip)
- Direct, high-Speed interface to IDT79R3000 Processor
- Supports for Coprocessor level IEEE P754 Floating-Point applications
- Floating-Point Registers: Stores 64-bit registers
- High-speed CEMOS technology
- Fully functional and software compatible with the MIPS Coprocessor level IDT79 R3010 FPA
- Military IDT/LG compliant to MIL-STD-883, Class B

DESCRIPTION:

The IDT79R3010 Floating-Point Accelerator (FPA) operates as a coprocessor for the IDT79R3000 Processor and extends the IDT79R3000 instruction set to perform arithmetic operations on floating-point formats.

Figure 1. IDT79R3010 Functional Block Diagram

IDT79R3010 FPA REGISTERS

The IDT79R3010 processor has sixteen 32-bit registers.

General Purpose Registers (FGR)

Figure 2. IDT79R3010 FPA Registers

- Floating-Point General Register (FGR)
- Floating-Point Control Register FCR
- Floating-Point Control Register FCR
- Floating-Point Register FPR

Load, Store, and Move Operations

Floating-Point General Registers (FGR)

There are 32 Floating-Point General Registers (FGR) on the FPA.

Floating-Point Registers (FPR)

Floating-Point Control Registers (FCR)

There are 2 Floating-Point Control Registers (FCR) on the FPA.

Floating-Point Operations

- Add
- Subtract
- Multiply
- Divide
- Absolute Value
- Move
- Negate
- Compare

Exception

COPROCESSOR OPERATION

- Load and Store

INSTRUCTION SET OVERVIEW

OP	Description
	Load/Store/Move Instructions
LWC1	Load Word to FPA
SWC1	Store Word from FPA
MTC1	Move Word to FPA
MFC1	Move Word from FPA
CTC1	Move Control Word to FPA
CFC1	Move Control Word from FPA
	Conversion Instructions
CVT.S.fmt	Floating-point Convert to Single FP
CVT.D.fmt	Floating-point Convert to Double FP
CVT.W.fmt	Floating-point Convert to Fixed-point

OP	Description
	Computational Instructions
ADD.fmt	Floating-point Add
SUB.fmt	Floating-point Subtract
MUL.fmt	Floating-point Multiply
DIV.fmt	Floating-point Divide
ABS.fmt	Floating-point Absolute Value
MOV.fmt	Floating-point Move
NEG.fmt	Floating-point Negate
	Compare Instructions
C.cond.fmt	Floating-point Compare

IDT79R3010 PIPELINE ARCHITECTURE

Instruction Execution

RISC CPU WRITE BUFFER

ADVANCE INFORMATION IDT79R2020

FEATURES

DESCRIPTION

Figure 1. The IDT79R2020 Write Buffer in the IDT79R2000 System.

MR0105702

PIN CONFIGURATION

PLCC

MIPS
R2000 Processor
Interface

78-00005 (C)

mips

MIPS
R2000 Processor
Interface

78-00005 (C)

MIPS R2000 Processor Interface

MIPS Computer Systems, Inc.
930 Arques Avenue
Sunnyvale, CA 94086
(408) 720-1700

MIPS CONFIDENTIAL

Tom Riordan

June 30, 1987

Table of Contents

This version of the R2000 Processor Interface corresponds to processor revisions 5.0 and above.

The differences between revision 5.0 and previous processor revisions are as follows:

(1) *The association of the processor's cache control outputs to the input clocks has changed to allow the design of faster systems. These differences are reflected throughout the document and are summarized in the table in Section 3, Cache Timing. An additional mode bit has been added to choose between the new clock bindings and the old clock bindings. How to select the desired mode is explained in the section Advanced Features.*

(2) *The quarter cycle maximum difference between the earliest and latest input clocks has been changed to a half cycle. This change also permits faster designs.*

(3) *A new output clock, CpSync*, has been added to provide a matched-loading synchronization input to coprocessors. This clock permits minimal offset in the phase lock circuitry.*

(4) *Additional information is provided to the coprocessors on the Exception* output. The coprocessors are now notified of the occurence of the fixup cycle so that there is no sensitivity to the electrical characteristics of the data bus during stalls.*

(5) *The mode inputs allowing seperate selection of the absence or presence of the instruction and data caches has been consolidated into a single mode selecting absence or presence of both.*

(6) *A mode input has been added which determines whether the data and tag buses are driven during coprocessor busy and write busy stalls. For designs that do not use the buses during these stalls, enabling the bus drive prevents the buses from floating for extended periods of time. This consideration can be important where high speed TTL logic inputs are attached to the bus as these inputs tend to oscillate if they float near the TTL logic trip point. While this revision of the processor is insensitive to these oscillations it could be an overall system design problem if buses are allowed to oscillate.*

1. Introduction

The R2000 processor supports interfaces to cache, main memory, and coprocessors. This document describes the connectivity and operation of each of these interfaces. Figure 1 illustrates a system which uses all three interfaces.

2. Operation Fundamentals

A *cycle* is the basic instruction processing unit of the R2000 processor. Cycles in which forward progress is made, that is, an instruction is retired, are called *run* cycles. An instruction is retired either by its completion or, in the presence of exceptions, its abortion. Cycles in which no forward progress is made are called *stall* cycles. Stall cycles are used for resolving exigencies such as cache misses on loads, write system busy during stores, and coprocessor interlocks. All cycles can be classified as either run cycles or stall cycles. Processor transactions which occur during the first half of a cycle are called *phase 1* transactions while those which occur during the second half are called *phase 2* transactions.

2.1. Run Cycle Operation

Run cycles are characterized by the unconditional transfer of an instruction into the processor during phase1 and the possible transfer of data either into or out of the processor during phase2. Whether or not a data transfer occurs, each run cycle is thought of as having an instruction-data, or ID, pair associated with it. The processor indicates that it is in the run state by assertion of the output signal Run*.

2.2. Stall Cycle Operation

Stall cycles are characterized by the processor maintaining a state consistent with resolving the stall while waiting for the stall condition to terminate. During the final cycle of a stall, that is, the cycle before reentering a run cycle, the ID pair which appeared or should have appeared during the last run cycle is placed on the data bus by the processor. This last stall cycle is used to restart the processor and coprocessor pipelines and in general to *fixup* the conditions which caused the stall. It is called the fixup cycle.

2.3. Processor Pipeline

The R2000 processor has a five stage pipeline and in general is simultaneously executing one pipeline stage for each of five instructions. The five pipeline stages are: *instruction fetch*, *register fetch*, *ALU*, *memory access*, and *writeback*. The pipeline stages are abbreviated as I, R, A, M, and W. The pipeline is illustrated in figure 2.

Figure 1. R2000 System Block Diagram

863 FH PG 0952

clock

phase | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |

Instr 0: | I | R | A | M | W |

Instr 1: | I | R | A | M | W |

Instr 2: | I | R | A | M | W |

Instr 3: | I | R | A | M | W |
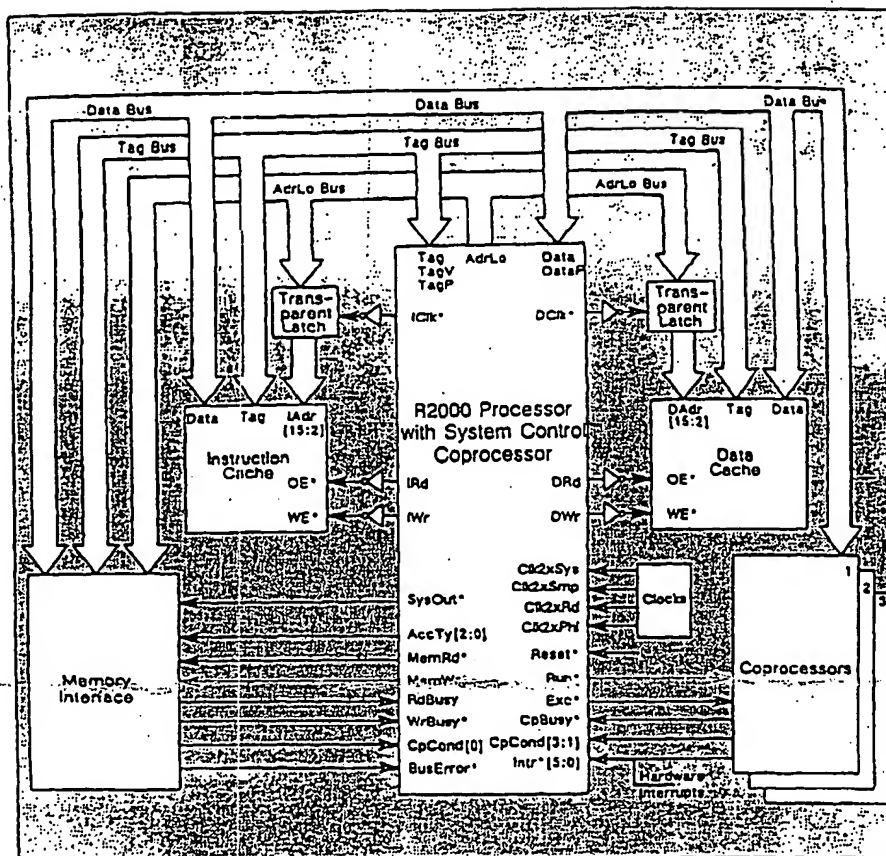
Instr 4: | I | R | A | M |
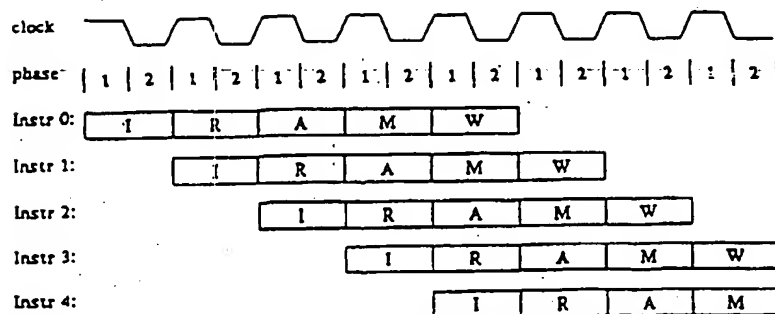
Figure 2: Processor Pipeline

For any particular instruction, the instruction itself is present on the data bus during phase 1 of the register fetch pipestage and the data transaction, if any, occurs during phase 2 of the memory access pipestage. The bus transactions relative to the processor pipeline are illustrated in figure 3.

clock

phase | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |

Instr 0: | I | R | A | M | W |

Data Bus: | IO | | DO |
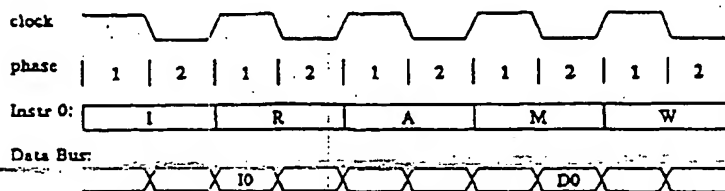
Figure 3: Bus transactions relative to pipeline

The bus transactions when the pipeline is full appear as follows:

clock

phase | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |

Data Bus: | I-1 | D-3 | IO | D-2 | I1 | D-1 | I2 | DO | I3 | DI |
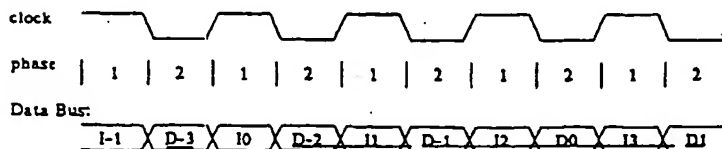
3. Cache Interface

The R2000 supports a split instruction-data cache; that is, it maintains separate caches for instructions and data. Each cache is direct mapped and each can range in size from 4K bytes to 64K bytes. The system in figure 1 showed a configuration with maximum size instruction and data caches.

3.1. Cache Format

Both the instruction and data caches have a line-size of one where each line contains 32 bits of data and 21 bits of tag. The tag consists of a single validity bit and a 20-bit page frame number. Additionally, each line contains 4 bits of parity for the data and 3 bits of parity for the tag. The format of a cache line is shown below.

59 57 56 55		36 35 32 31		0
TagP V	PFN	DataP	Data	
3 1	20	4	32	
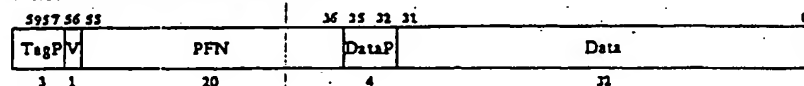
where:

Data	is the cache data
DataP	is parity over the Data field
PFN	is the Page Frame Number
V	is the valid bit
TagP	is parity over the V and PFN fields

DataP(0) contains parity over Data(7:0), DataP(1) contains parity over Data(15:8), DataP(2) contains parity over Data(23:16), and DataP(3) contains parity over Data(31:24). Parity over the data plus the parity bit is even.

3.2. Cache Operation

The caches are addressed by the 16-bit address bus, AdrLo(15:0). Since AdrLo presents byte addresses and the caches are organized as words, its least significant two bits are not used. The most significant four bits of AdrLo are identical to the least significant four bits of the Tag but are output with AdrLo timing. This overlap allows cache size to vary with implementation. The table below summarizes the use of AdrLo for all possible cache sizes.

cache size (bytes)	cache size (words)	AdrLo
4 kb	1024	11:2
8 kb	2048	12:2
16 kb	4096	13:2
32 kb	8192	14:2
64 kb	16384	15:2

During each run cycle it is possible for both an instruction and data cache reference to occur with the references offset from one another by a phase. Instruction references begin their reference during phase2 and transfer data during the following phase1 while data references begin during phase1 and transfer data during phase2. Figure 4 illustrates the operation of the cache interface for an execute-load-store-load sequence. For instructions, data is always transferred from the cache to the processor, while for data, the direction of transfer depends on whether the operation is a load or a store. In addition to the processor signals, the figure shows the local instruction and data cache address buses, IAdr and DAdr, respectively. These buses are latched versions of the processor address bus which are created using transparent latches controlled by IClk* and DClk*.

During run cycles the access type bus. AccTyp(2:0), indicates whether or not a phase 2 transaction is scheduled for that cycle and the size of the datum being transferred. AccTyp(1:0) encodes the size of the transaction. The encoding is illustrated in the section describing main memory reads. AccTyp(2) indicates that no data transaction is occuring during the current cycle.

instr	Execute		Load		Store		Load	
cycle	Run		Run		Run		Run	
phase	1	2	1	2	1	2	1	2
Data:	I	Unused	I	Din	I	Dout	I	Din
Tag:	ITag	Unused	ITag	DTagIn	ITag	DTagOut	ITag	DTagIn
AdrLo:	Unused	IAdr	DAdr	IAdr	DAdr	IAdr	DAdr	IAdr

AccTyp(1:0):

	Unused		Size		Size		Size	

AccTyp(2):

LAdr:	IAdr		IAdr		IAdr		IAdr		IAdr	
DAdr:	Unused		DAdr		DAdr		DAdr	

ICIk*

DCIk*

IRd

DRd

IWr

DWr
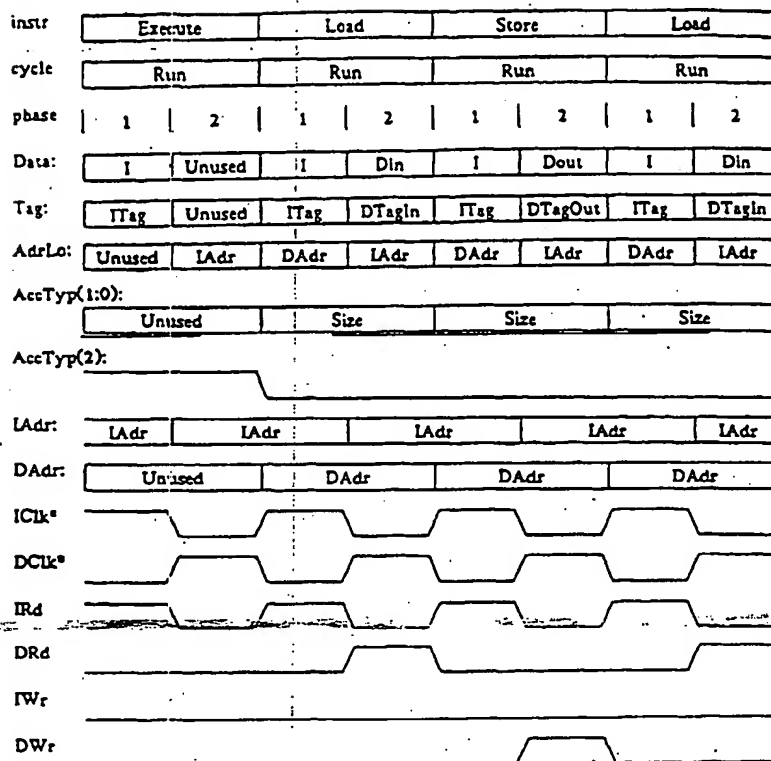
Figure 4: Cache Operation

3.3. Cache Timing

The processor has four separate double frequency input clocks. The differences between these clocks are used to position the cache control signals for optimal performance under a wide variety of conditions. Note that the absolute timing of these clocks with respect to the processor outputs is undefined: only the differences are important. The four clocks and their use are described below.

(1) Clk2xSys: Clk2xSys is the master clock and must lead all the others. Clk2xSys determines the position of SysOut* with respect to the data, tag, and address buses. It is positioned so

that 1) the data, tag, and address buses maintain the desired setup and hold time with respect to a buffered version of SysOut* and 2) inputs which are clocked by a buffered SysOut* meet the processors setup and hold requirements. Clk2xSys also terminates IRd and DRd to prevent cache read to cache read bus contention.

(2) Clk2xSmp: This clock determines the sample point for data coming into the processor on all processor inputs except for those coming directly from coprocessors. It is positioned so that the data is available for latching by the internal phase clocks.

(3) Clk2xRd: Clk2xRd 1) controls the output enable for the cache RAMs, 2) disables the drive of the data and tag buses, and 3) determines the assertion edge of the address latch clocks. IClk* and DClk*. It is positioned to maximize output enable time and to provide sufficient address access to sample, address hold from end of write, and data hold from end of write.

(4) Clk2xPhi: Clk2xPhi determines the position of the internal phases, phase1 and phase2. The data, tag, and address buses are driven with respect to Clk2xPhi.

The table below summarizes the 2xClock dependency of the processors timing controlled outputs. Outputs are referenced only to rising edges of the 2xClocks. The assertion dependency is indicated by ↑ and the deassertion dependency is indicated by ↓.

	Clk2xSys	Clk2xSmp	Clk2xRd	Clk2xPhi
IClk*,DClk*		↓	↑	
IRd,DRd	↓		↑	
IWr,DWr		↑ ↓		
SysOut*	↑ ↓			
Data,Tag			↓	↑
Address				↑ ↓
All Others				↑ ↓

In the timing diagrams which follow, timing specifications are given relative to a shifted version of the processor output clock SysOut*. The shift amount is equal to the Clk2xSys to Clk2xPhi delta as established by the input 2x clocks. As shown in figure 5, the Clk2xSys to Clk2xPhi delay is defined as TSys.
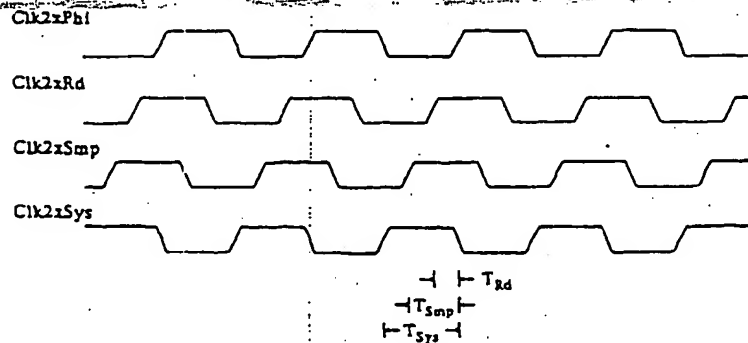
Figure 5: 2x Input Clocks

The shifted version of SysOut* is called PhiOut* and even though the processor does not actually produce this output it is shown on the timing diagrams for reasons of clarity.

QED 00013
CONFIDENTIAL

SysOut* is produced rather than PhiOut* since this provides a signal with timing appropriate for synchronizing system transactions to the processor. Timings are given relative to PhiOut* since this makes determining the position of the input clocks the most straightforward. Note that the timing of any output with respect to SysOut* can be determined from its timing with respect to PhiOut* by adding TSys.

Detailed timing for a store-load sequence is shown in figure 6.

863 FH PG 0958

Figure 6: Cache Timing

863 FH PG 0959
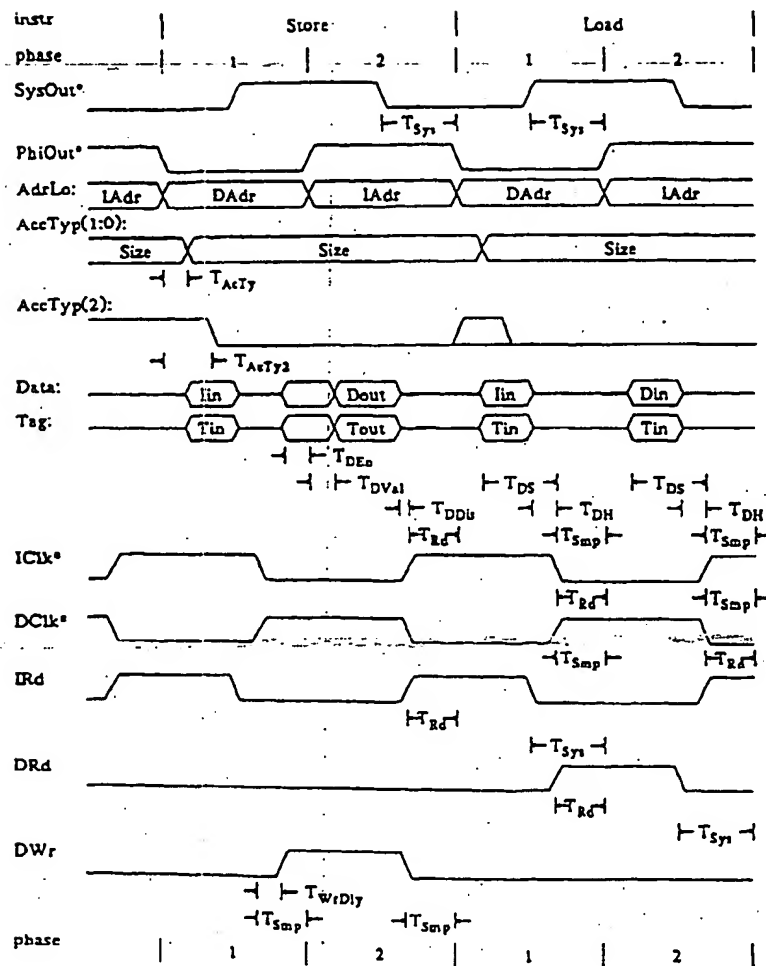
4. Main Memory Interface

The principal supporting mechanism for main memory operations is the processor stall cycle. Main memory stalls occur when loads miss in the cache or when stores are blocked by the write system. The minimum processor stall for both read and write stalls consists of a single stall cycle and a fixup cycle as illustrated in figure 7. There is no maximum length for a stall.

instr	Lead		None		None		Execute	
cycle	Run		Stall		Fixup		Run	
phase	1	2	1	2	1	2	1	2
Data:	I	D	iI	MemoryIn	I	D	I	HiZ
Tag:	ITag	DTag	ITag	RdAddrHi	ITag	DTag	ITag	HiZ
AdrLo:	DAdr	IAdr	RdAddrLo		DAdr	IAdr	DAdr	IAdr
MemRd*								
RdBusy								
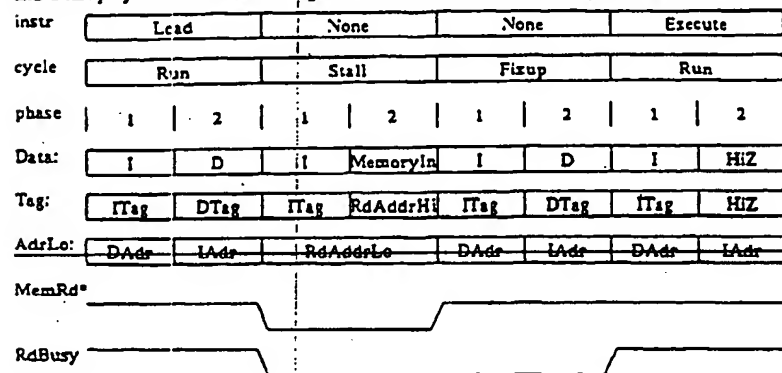
Figure 7: Minimum read/write stall

4.1. Main Memory Reads

When a load misses in the cache, a main memory read is initiated. Misses occur when (1) the valid bit is not set (2) the tags are not equal (3) a parity error is detected or (4) the reference is uncached. Main memory reads are supported by *read busy* stalls and the MemRd*, RdBusy signal pair. Figures 8 and 9 illustrate the read busy stalls for a data cache miss and an instruction cache miss, respectively. Entry into the stall is indicated by the assertion of MemRd* and occurs on the cycle following the one in which the reference missed. During the stall, the processor presents the read address on the Tag and AdrLo buses and tristates the Data bus. The processor maintains these conditions until RdBusy is deasserted. In order to maintain a read busy stall, the memory system must assert RdBusy no later than phase1 of the cycle in which MemRd* was asserted. To terminate a read busy stall the memory system deasserts RdBusy during phase1 of the cycle in which it will place valid data on the Data bus. The cycle following that in which RdBusy is deasserted is a fixup cycle. During this cycle the appropriate cache, either instruction or data, is written with the data returned by main memory. The processor does not require the memory system to provide correct parity for the returned data. Simultaneous with the data, the generated data parity, tag, and tag parity are also written. The cache write does not occur if the stall was due to an uncached reference. The processor resumes run operation on the cycle following the fixup.

During all instruction cache misses and during data cache misses for cached references, the least significant two bits of Access Type always indicate a word reference. For uncached data references the access type bits indicate the actual size of the reference as indicated by the table below.

AccessType(1:0)	size
00	byte
01	half word
10	tribyte
11	word

The most significant bit of access type indicates whether the stall is for an instruction or a data cache miss.

instr	Load		None		None		None		None		Execute	
cycle	Run		Stall		Stall		Stall		Fixup		Run	
phase	1	2	1	2	1	2	1	2	1	2	1	2
Data:	I	BadD	I		MainMemoryIn				I	GoodD	I	HiZ
Tag:	ITag	BadT	ITag		ReadAddress(31:16)				ITag	GoodT	ITag	HiZ
AdrLo:	DAdr	IAdr			ReadAddress(15:0)				DAdr	IAdr	DAdr	IAdr

AccTyp(1:0):

	DAcTy				DAcTy				DAcTy		DAcTy	

AccTyp(2)

MemRd*

RdBusy

DWr

Figure 8: Data Cache Miss

863 FH PG 0962

instr	Execute	None	None	None	None	Execute		
cycle	Run	Stall	Stall	Stall	Fixup	Run		
phase	1 | 2	1 | 2	1 | 2	1 | 2	1 | 2	1 | 2		
Data:	BadI | D	I	MainMemoryIn			GoodI | D	I | HiZ
Tag:	BadT | DTag | ITag	ReadAddress(31:16)			GoodT | DTag | ITag | HiZ
AdrLo:	DAdr | IAdr	ReadAddress(15:0)			DAdr | IAdr | DAdr | IAdr

AccTyp(1:0):

	DAcTy	Word			DAcTy	DAcTy

AccTyp(2)

MemRd*

RdBusy

IWr
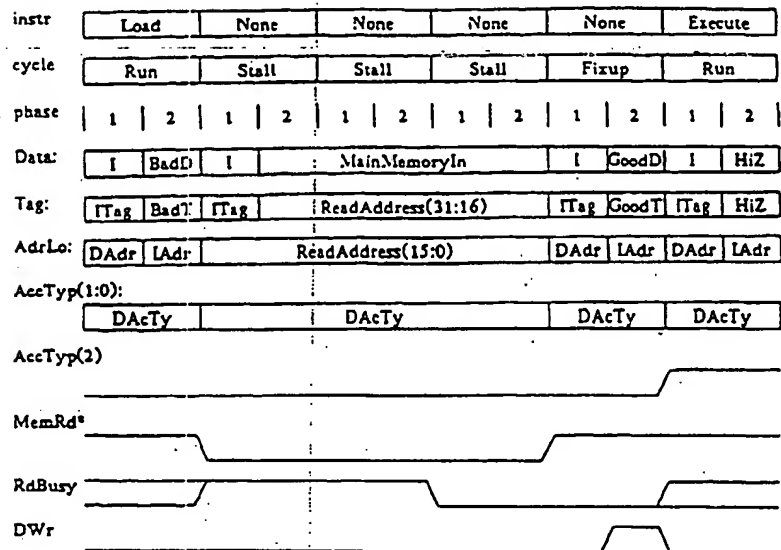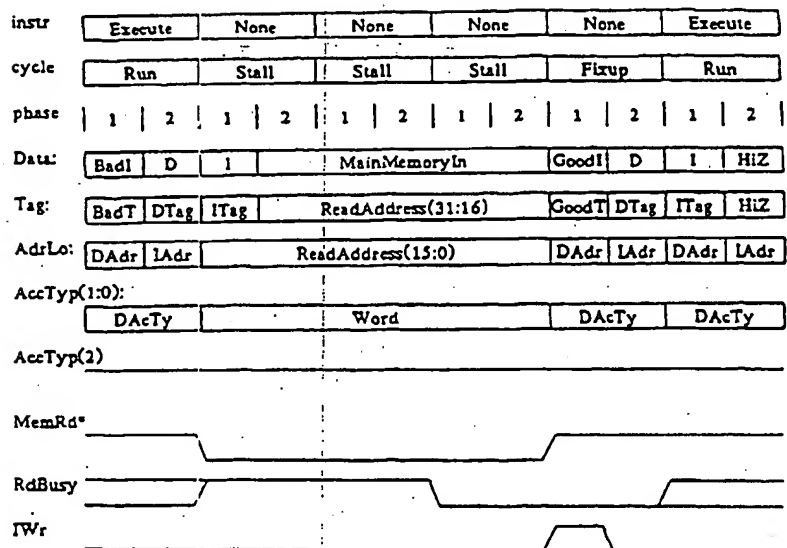
Figure 9: Instruction Cache Miss

4.2. Read Timing

Timing for the beginning and end of a read busy stall is illustrated in figures 10a and 10b respectively. Note that in order to maintain maximum execution rate, the processor uses phase1 of the first stall cycle to accomplish the transition between run and stall operation for the tag and data buses. MemRd* is asserted with respect to phase1, however, in order to notify the memory system as quickly as possible of the impending read. The timing for both IWr and DWr is shown during the fixup cycle while in practice only one or the other will occur depending on the type of stall.
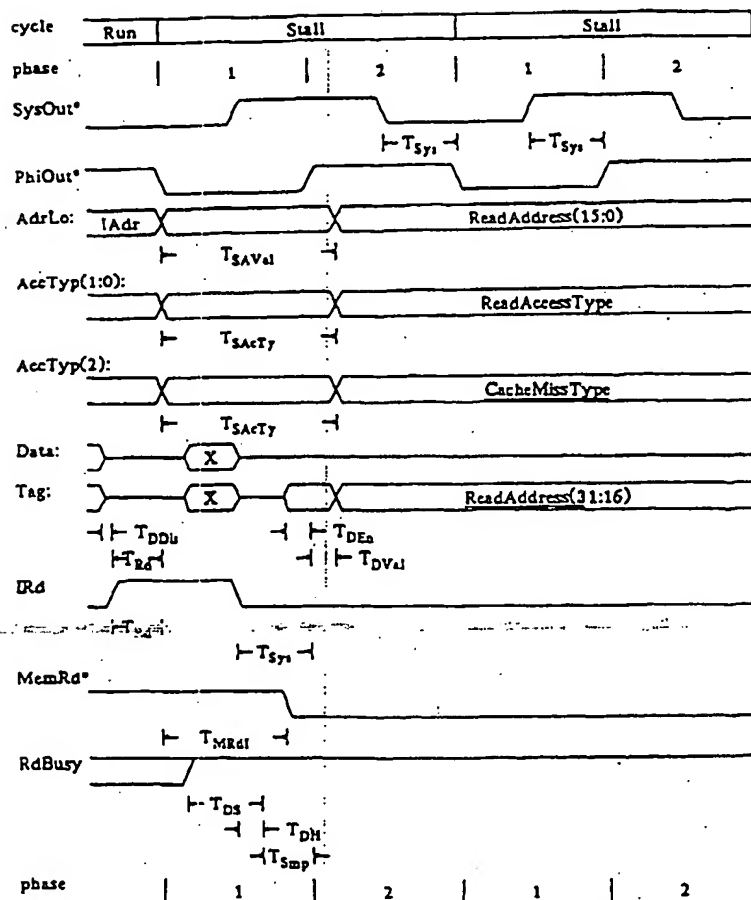
Figure 10a: Read Busy Timing - Beginning of Stall
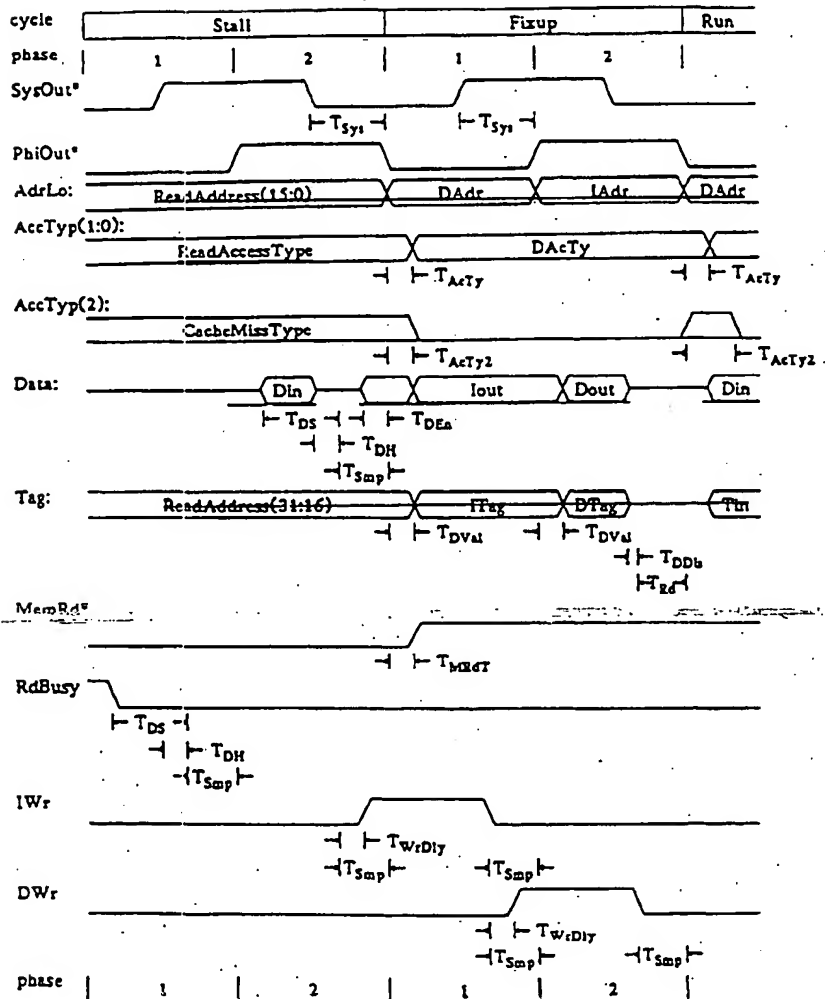
Figure 10b: Read Busy Timing - End of Stall

4.3. Main Memory Writes

Main memory writes are accomplished through a write buffer which is presumed to accept writes at cache speeds. The occurence of a write is indicated to the write buffer by the assertion of MemWr*. If the write buffer becomes full it can cause a further write attempt to result in a *write busy* stall as illustrated in figure 11. The write which occurs during the first cycle shown fills the write buffer causing it to assert WrBusy*. This assertion must occur before the end of the next cycle. The write which is attempted in the second cycle is not accepted by the write buffer and is redone by the processor during the fixup cycle of the stall. The write busy stall is maintained until the write buffer deasserts WrBusy* indicating that it is now ready to accept a write. The cycle following its deassertion will be the fixup cycle.

instr	Store	Store	None	None	None	None
cycle	Run	Run	Stall	Stall	Stall	Fixup

phase | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |

| Data: | I | D | I | D | - | Unused | | | | I | D |

| Tag: | ITag | DTag | ITag | DTag | - | Unused | | | | ITag | DTag |

| AdrLo: | DAdr | IAdr | DAdr | IAdr | WriteAddress(15:0) | | | | DAdr | IAdr |

AccTyp(1:0):

| | DAcTy | DAcTy | DAcTy | | | DAcTy |

MemWr*

WrBusy*

DWr
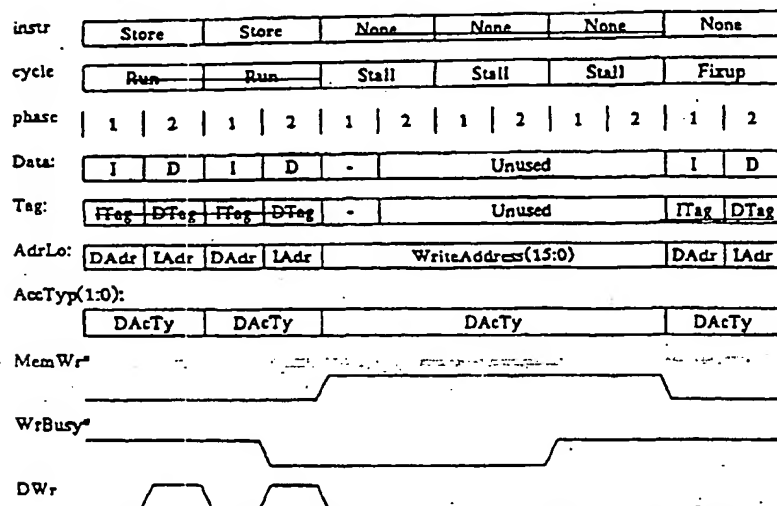
Figure 11: Write Busy Stall

During cycles in which MemWr* is asserted as well as during write busy stalls, Access Type indicates the size of the transaction as indicated in the table below.

AccessType(1:0)	size
00	byte
01	half word
10	tribyte
11	word

4.4. Write Timing

Timing for the beginning and end of a memory write followed by a write busy stall is illustrated in figures 12a and 12b, respectively.
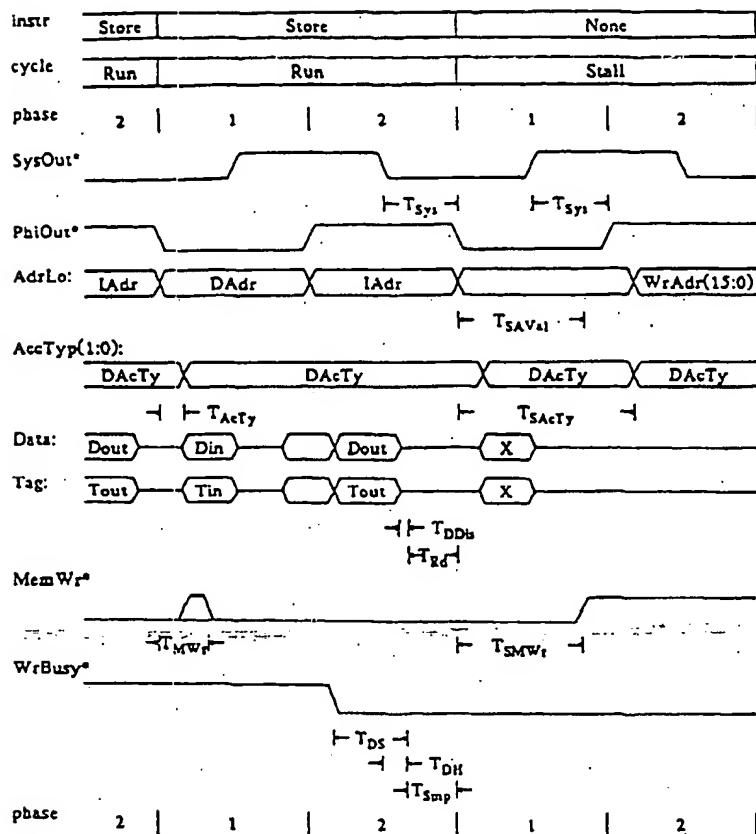
Figure 12a: Write Busy Stall - Memory Write and Beginning of Stall
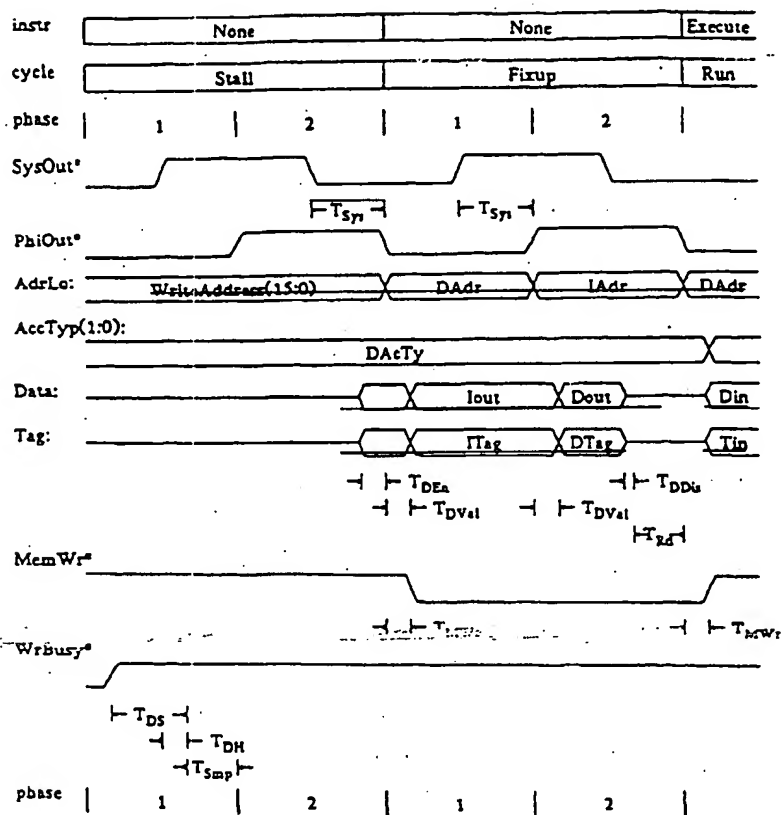
Figure 12b: Write Busy Stall - End of Stall

4.5. Bus Error

The occurence of a extraordinary failure during either a read busy or write busy stall is signalled to the processor by BusError*. BusError* can be asserted either before or concurrent with the deassertion of the normal stall termination signals, RdBusy or WrBusy*. If asserted before the normal terminators the effect is as if the terminators were also deasserted. Stalls terminated by BusError* are subject to retry in the same manner as when terminated by deassertion of the appropriate Busy. That is, if RdBusy or WrBusy* is reasserted during the fixup cycle of the stall then a retry will occur. See the section on retry for more details. On a successful retry the effect will be as if BusError* had not been asserted. A successful retry in this instance is defined to be one in which BusError* is not reasserted. During the fixup cycle of a bus error terminated stall, the appropriate cache location is invalidated by turning off the valid bit before the write. Correct parity is maintained by inverting the sense of the most significant Tag bit. Termination of a read busy and write busy stall by BusError* is shown in the figures 13a and 13b. The data setup requirements for BusError* are identical to the deassertion requirements of RdBusy and WrBusy*. Note finally that BusError* is only sampled during read or write busy stalls.
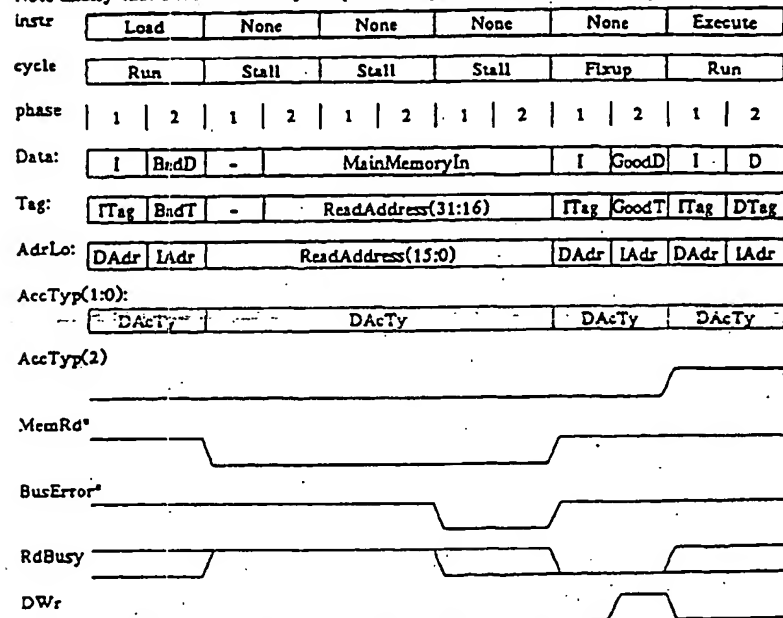
Figure 13a: Read Busy stall terminated by a Bus Error

instr	Store	Store	None	None	None	None
cycle	Run	Run	Stall	Stall	Stall	Fixup
phase	1 | 2	1 | 2	1 | 2	1 | 2	1 | 2	1 | 2
Data:	I | D	I | D	-	Unused		I | D
Tag:	ITag | DTag	ITag | DTag	-	Unused		ITag | DTag
AdrLo:	DAdr | IAdr	DAdr | IAdr	WriteAddress(15:0)			DAdr | IAdr

AccTyp(1:0):

	DAcTy	DAcTy	DAcTy			DAcTy

MemWr*

BusError*

WrBusy*

DWr
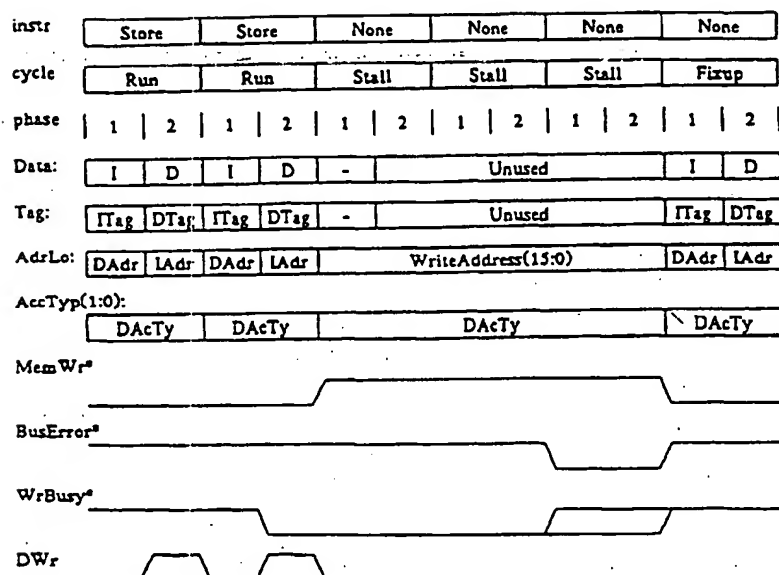
Figure 13b: Write Busy Stall terminated by a Bus Error

5. Coprocessor Interface

The R2000 supports a tightly coupled coprocessor interface: all coprocessors maintain synchrony with the main processor, reside on the same data bus as the main processor, and participate in bus transactions in an identical fashion to the main processor.

The interface supports up to four separate coprocessors. Coprocessor 0 - the system coprocessor - is contained within the main processor. Coprocessor 1 is the floating-point coprocessor. Coprocessors 2 and 3 are undefined at present.

5.1. Coprocessor Operation Fundamentals

During each cycle in which a valid instruction-data pair is on the data bus, the coprocessors accept an instruction. The coprocessors decode the instruction in parallel with the main processor and, if it is a coprocessor instruction, one of the coprocessors will proceed to execute the instruction. The setup and hold requirements for instruction transfers to the coprocessor are identical to those of the processor.

The coprocessors maintain synchronization with the main processor by monitoring the signals Run* and Exception*. Run* is asserted by the processor during run cycles and deasserted during stall cycles. When the processor deasserts Run*, the coprocessors disregard the instruction-data pair presented during the last cycle. When Run* is reasserted, the coprocessors take, as replacement for the instruction-data pair which was disregarded, the instruction-data pair from the previous cycle - the previous cycle having been the fixup cycle for whatever stall was occuring.

Exception* is used by the processor to transmit four independent pieces of information to the coprocessors.

(1) During phase 1 of run cycles Exception* indicates whether an exception has occured for the instruction which is currently in its *writeback* pipestage. Unless the exception is occuring as a result of an interrupt request by the coprocessor, the assertion of Exception* prevents any state from being committed in the coprocessor.

(2) During phase 2 of run cycles Exception* indicates whether an interrupt request is being granted for the instruction which is currently in its *memory access* pipestage. When an exception occurs corresponding to the granting of an interrupt request, the state indicating the type of exception is committed within the coprocessor.

(3) During phase 1 of stall cycles Exception* indicates whether the current stall cycle is a fixup cycle. When a fixup cycle is occuring, it is guaranteed that the data present on the data bus is electrically valid.

(4) Finally, during phase 2 of stall cycles, Exception* indicates whether the current stall is a Coprocessor Busy stall. The Coprocessor Busy stall indication in combination with the CpBusy* input can be used by entities external to the processor to gain access to the caches. The information content of the Exception* line is summarized in the table below.

	phase 1	phase 2
Run	Exc1W*	IntGr2M*
Stall	Fixup1*	CPBusy2*

5.2. Coprocessor Instructions

The interface supports three types of coprocessor instructions: loads/stores, operations, and processor-coprocessor transfers. Each type is described below in terms of its demands on the interface. Timing of the coprocessor interface during run is illustrated in figure 14.

QED 00028
CONFIDENTIAL
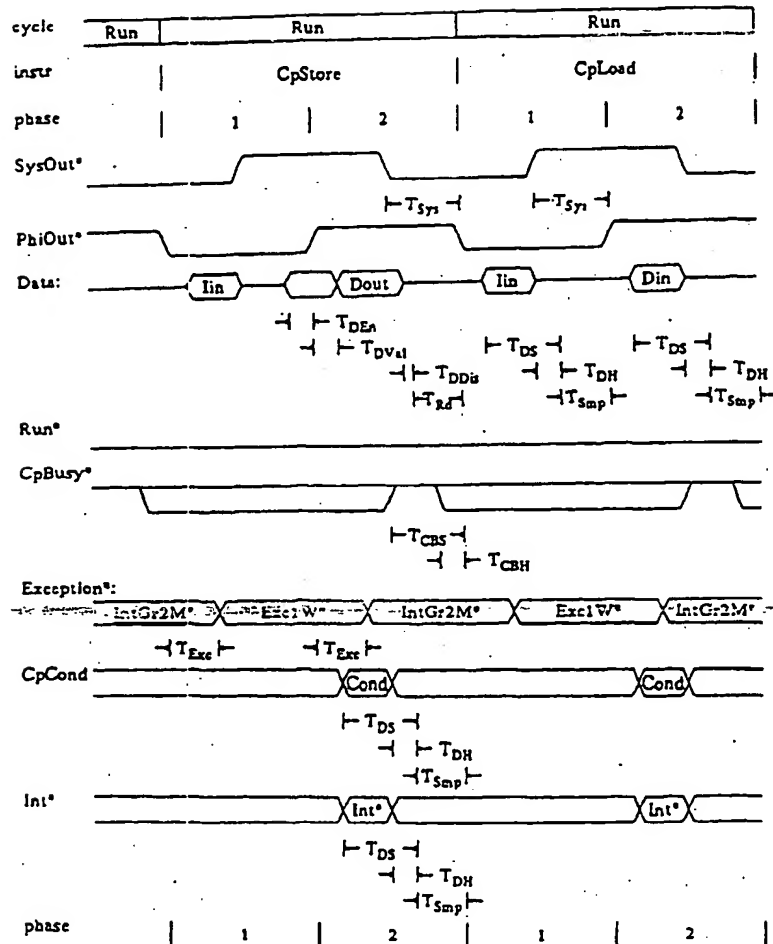
Figure 14: Coprocessor Interface Timing

5.2.1. Coprocessor Loads/Stores

During run cycles, the operation and timing of coprocessor loads and stores is identical to that of the main processor. On a load the coprocessor will be accepting data off the bus and on a store it will be driving the bus. On loads the main processor also reads in the data and tag for purposes of miss detection. On stores the coprocessor must generate data parity. All address generation, cache, and memory control functions are provided by the main processor.

During all stall and fixup cycles, the coprocessors are passive; if a coprocessor store is blocked by a write busy stall or if the cycle in which the coprocessor store occurs is redone due to any other stall, the main processor will re-present the coprocessor data during the stall's fixup cycle.

5.2.2. Coprocessor Operations

Coprocessor operations occur within the coprocessors and only affect the interface when they change the coprocessor condition output or cause stalls or exceptions. Coprocessors stalls and .exceptions are described separately below.

5.2.2.1. Coprocessor Conditions

Each coprocessor has a condition input into the main processor called CpCond(n). The coprocessor condition lines are sampled by the processor during phase2 of every run cycle. Figure 14 illustrates the timing requirements of the CpCond inputs. If the processor executes a coprocessor branch instruction, the state of the appropriate CpCond input determines the direction of the branch.

5.2.3. Coprocessor - Processor Transfers

Coprocessor-processor transfers have identical input and output characteristics as loads and stores; that is, for a processor to coprocessor transfer the processor drives the data bus as for a store and the coprocessor inputs from the bus as for a load. For coprocessor to processor transfers the roles are reversed. Parity is not checked for either direction of transfer.

5.3. Coprocessor Stalls

To provide synchronization when required, the processor supports *coprocessor busy* stalls. The operation of such a stall is illustrated in the figure 15. To initiate a coprocessor busy stall, the coprocessor must assert CpBusy* during the *ALU* cycle of the coprocessor instruction. To terminate the stall CpBusy* must be deasserted during phase1. The cycle following that in which CpBusy* is deasserted will be the fixup cycle.

instr	Execute	None	None	None	Execute
cycle	Run	Stall	Stall	Fixup	Run
phase	1 | 2	1 | 2	1 | 2	1 | 2	1 | 2
Data:	I | D	I | Unused		I | D	I | D
Tag:	ITag | DTag	ITag | Unused		ITag | DTag	ITag | DTag
AdrLo:	DAdr | IAdr	DataReadAddress(15:0)		DAdr | IAdr	DAdr | IAdr

AccTyp(1:0):

DAcTy	DAcTy	DAcTy	DAcTy

Exception*:

Exc* | IntGr*	Fixup* | CPBsy*	Fixup* | CPBsy*	Fixup* | CPBsy*	Exc* | IntGr*

CpBusy*

Run*
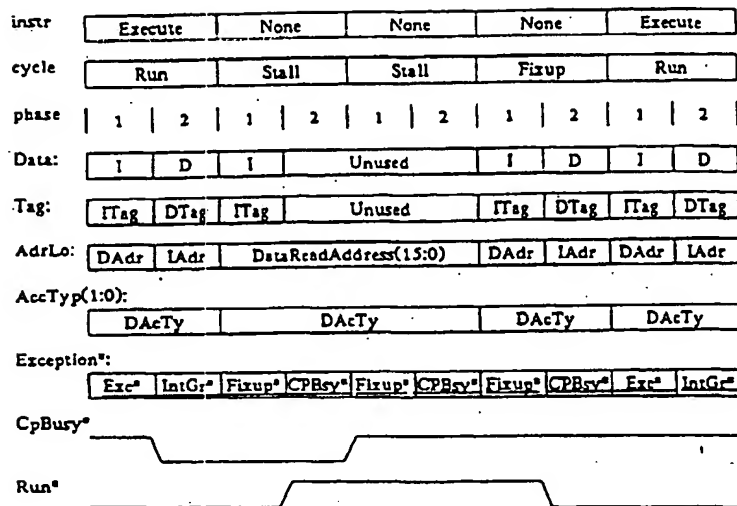
Figure 15: Coprocessor Busy Stall

Timing for the beginning and end of a coprocessor busy stall is shown in figures 16a and 16b. respectively. Note that CpBusy* has different setup and hold requirements than other processor inputs: CpBusy* is presumed to be coming from a tightly coupled coprocessor and its timing is directly related to the internal phase clocks.

863 FH PG 0975
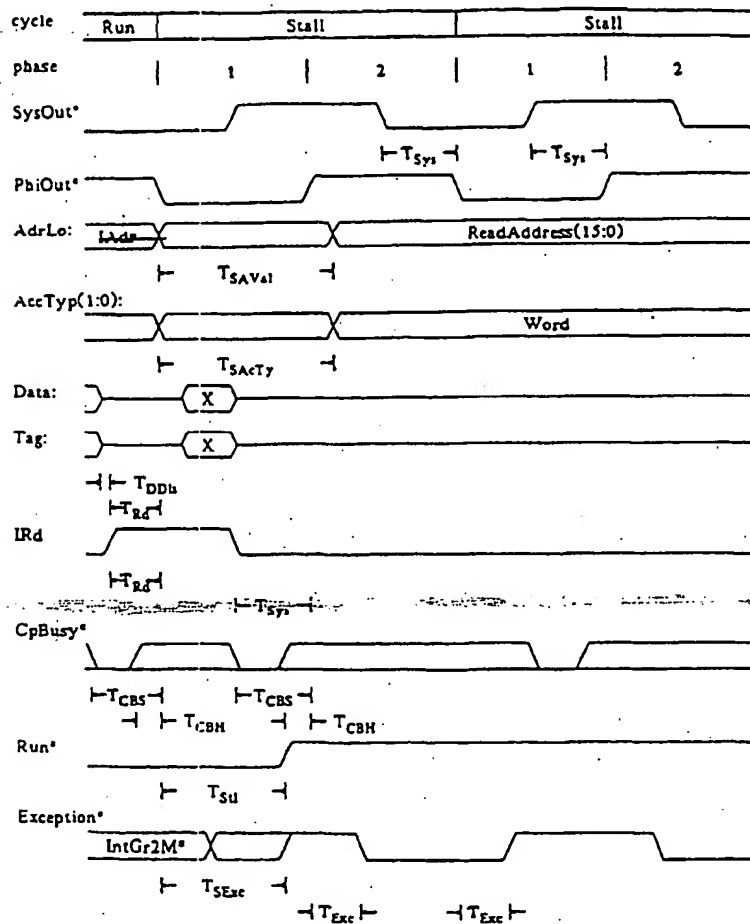
Figure 16a: Coprocessor Busy Timing - Beginning of Stall

863 FH PG 0976
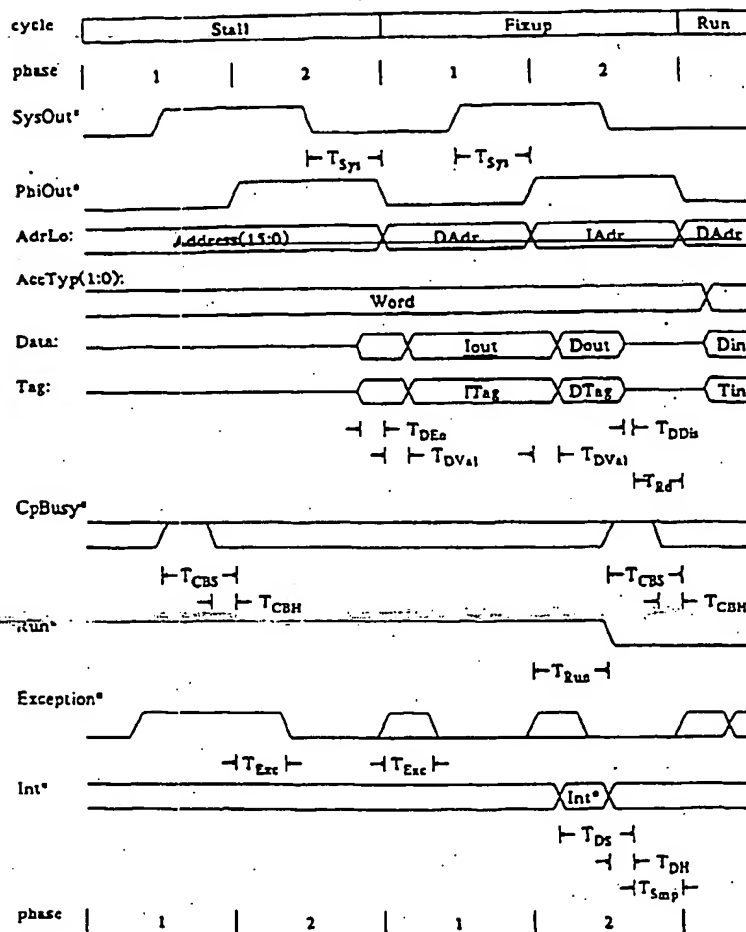
Figure 16b: Coprocessor Busy Timing - End of Stall

5.3.1. Coprocessor Busy Retry

Coprocessor busy stalls can be reinitiated by reasserting CpBusy* during the fixup cycle of the stall. However, unlike the retry for a read or write stall, a coprocessor busy retry may or may not be granted. Specifically, if an interrupt occurs during the initial stall then the retry will not be granted as the interrupt will abort the instruction which is requesting the stall. Figure 17 illustrates the timing of a coprocessor busy retry.

863 FH PG 0978
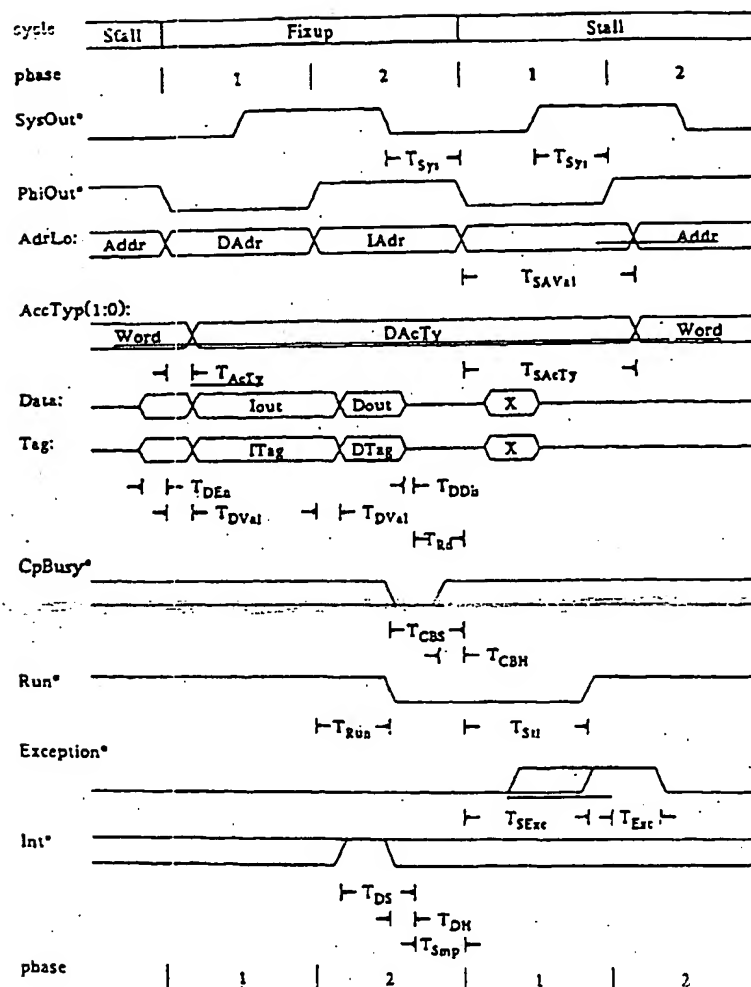
Figure 17: Coprocessor Busy Retry Timing

5.4. Coprocessor Exceptions

Coprocessors signal exceptions to the main processor through one of the processors interrupt inputs. The interrupt inputs are sampled during phase 2 of all run cycles and during the final fixup cycle of a stall sequence. Signalling precise exceptions via the interrupt inputs requires that the interrupt be asserted during the *ALU* pipestage of the instruction causing the exception. When signalled precisely, the processor signals interrupt grant back to the coprocessor during the *memory access* pipestage. The timing of the interrupt input is indicated in figures 14 and 16.

5.5. Processor-Coprocessor Synchronization

To operate the processor system at maximum speed requires that that there be minimum skew between the processor and coprocessors. To facilitate the deskewing of the processor and coprocessors, the processor provides a fixed *phase delay* in its input clock paths over and above the delay which is introduced naturally in the process of clock buffering. The phase delay is approximately equal to the expected worst case part to part variation in SysOut* under nominal operating conditions. The coprocessors contain a variable delay in their input clock paths which is set dynamically by comparing their output clock to the processors CpSync* output. CpSync* is nominally identical to SysOut* and is provided specifically for processor-coprocessor synchronization. The output clock of the coprocessor is loaded in a similar fashion to CpSync* of the processor to maximize matching. The additional phase delay path on the processor is enabled by asserting Int(4)* during reset. When disabled, the Clk2xSys to SysOut* delay will take on its nominal value. Figure 18 illustrates the qualitative effect of enabling the processor phase delay.
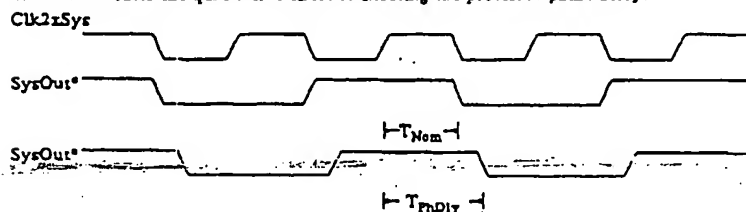
Figure 18: Phase Delay Effect

6. Internal Stalls

There are two types of stalls which the processor can enter which require no terminating action on the part of the interface: *mult / div busy* stalls and *microtlb fill* stalls. These are internally initiated stalls whose only indication of occurence is the deassertion of Run*.

Mult/div busy stalls occur on an attempt to read the result registers of the integer multiply/divide unit while a multiply or divide is in progress. Externally, the mult/div busy stall appears identical to a coprocessor busy stall: an internal MDBusy signal initiates and terminates the stall.

Microtlb fill stalls occur when an instruction translation misses in the instruction TLB cache. When such a miss occurs the processor takes a single cycle stall to refill the microtlb from the main TLB. Since the stall is only one cycle it is of necessity a fixup cycle. Moreover, since the processor is going directly from a run cycle into a fixup cycle, both the deassertion and reassertion of Run* are governed by TRun.

Figure 19 illustrates a mult/div busy and a microtlb fill stall.
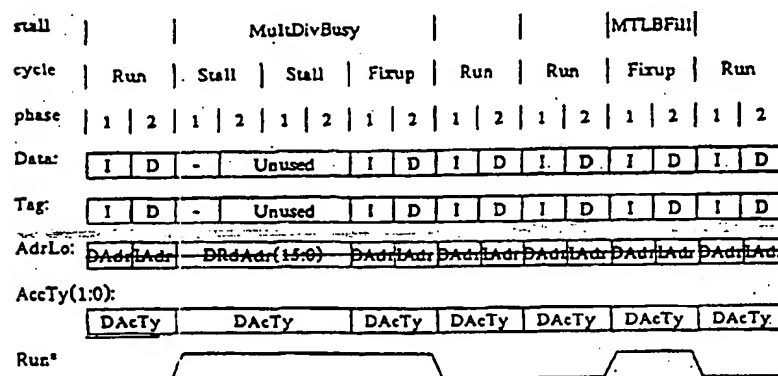
Figure 19: Mul/Div Busy and MicroTLB Fill Stall

7. Multiple Stalls

Multiple stalls are possible whenever more than one stall initiating events occur within a single run cycle. An example is a cycle containing a load where both the instruction and data reference miss in the cache. The most important characteristic of any multiple stall sequence is the validity of the instruction-data pair presented on the data bus during the final fixup cycle. The operation of the two most common multiple stalls is illustrated in figures 20a and 20b. Figure 20a illustrates a data cache miss followed by an instruction cache miss and figure 20b illustrates a write busy followed by an instruction cache miss.
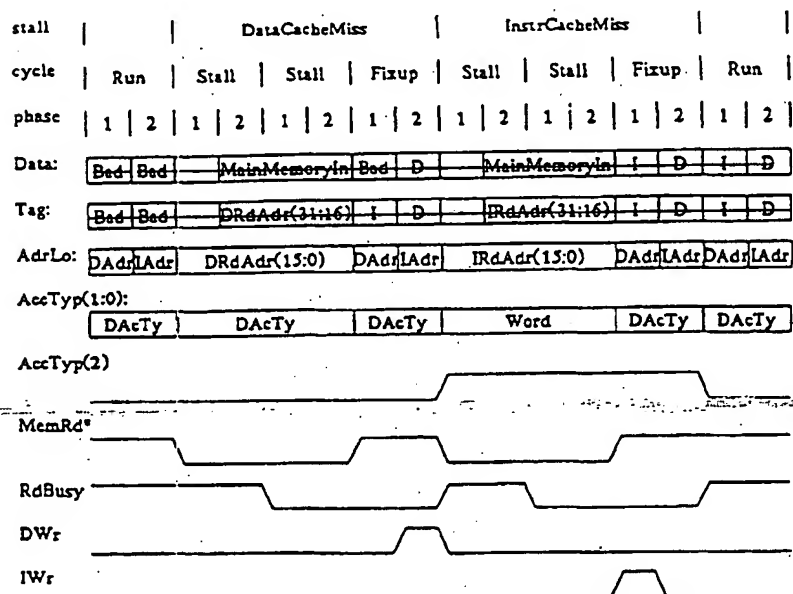
Figure 20a: Data Cache Miss - Instruction Cache Miss

Internal Stalls

stall			WriteBusy			InstrCacheMiss			
cycle	Run	Stall	Stall	Fixup	Stall	Stall	Fixup	Run	
phase	1 | 2	1 | 2	1 | 2	1 | 2	1 | 2	1 | 2	1 | 2	1 | 2	
Data:	Bad | D | -	Unused	Bad | D | -	MainMemoryIn | I | D | I | D
Tag:	Bad | D | -	Unused	I | D | -	IRdAdr(31:16) | I | D | I | D
AdrLo:	DAdr|IAdr	DRdAdr(15:0)	DAdr|IAdr	IRdAdr(15:0) | DAdr|IAdr | DAdr|IAdr
AccTyp(1:0):									
	DAcTy	DAcTy	DAcTy	Word	DAcTy | DAcTy

AccTyp(2)

MemWr*

WrBusy*

MemRd*

RdBusy

DWr

IWr
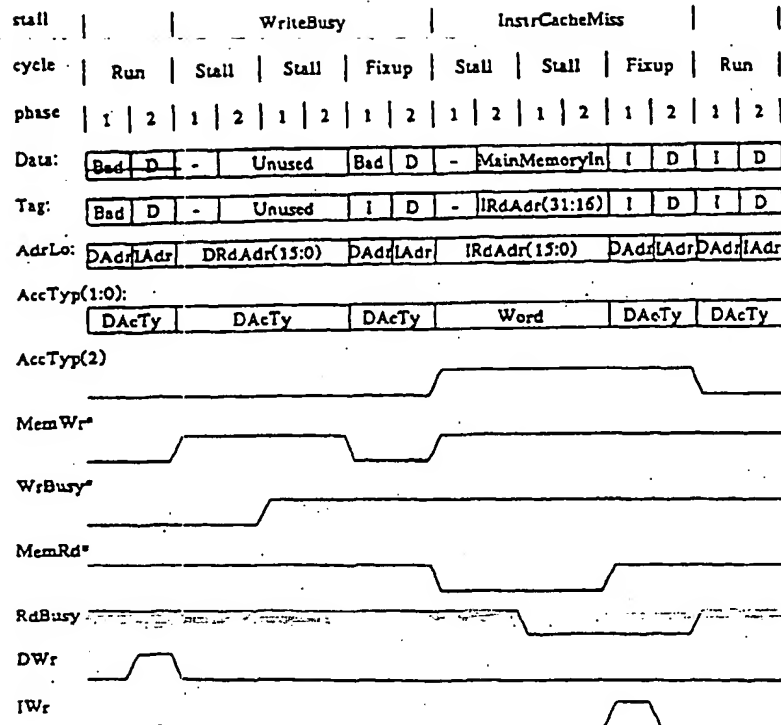
Figure 20b: Write Busy - Instruction Cache Miss

For the general case of multiple stalls. the service order is given below.

(1) Data Cache Miss or Write Busy - These are mutually exclusive as one occurs due to a load and the other due to a store.

(2) Coprocessor Busy

(3) Instruction Cache Miss

(4) Multiplier/Divider Busy

(5) MicroTLB Miss

For stalls which can be resolved without main processor intervention. such as coprocessor busy stalls. the stall initiator/terminator signals are sampled every cycle. If. while servicing another stall. the initiator for a self resolving stall is deasserted then no stall will occur.

863 FH PG 0983

8. Retry

Retry is a mechanism for redoing a stall that has already received its stall termination signal. For read stalls. retry allows error detection/correction to occur in parallel with data transfer. Figure 21 illustrates the operation of retry for the case of a data cache miss. In general. to retry the stall. the stall terminator - RdBusy. WrBusy. or CpBusy - is reasserted during the fixup cycle. From that point on. the retry stall is indistinguishable from the original stall. For read busy and write busy stalls. the retry is guaranteed to occur if requested. Further details concerning the details of coprocessor busy retry can be found in the section describing the coprocessor interface.
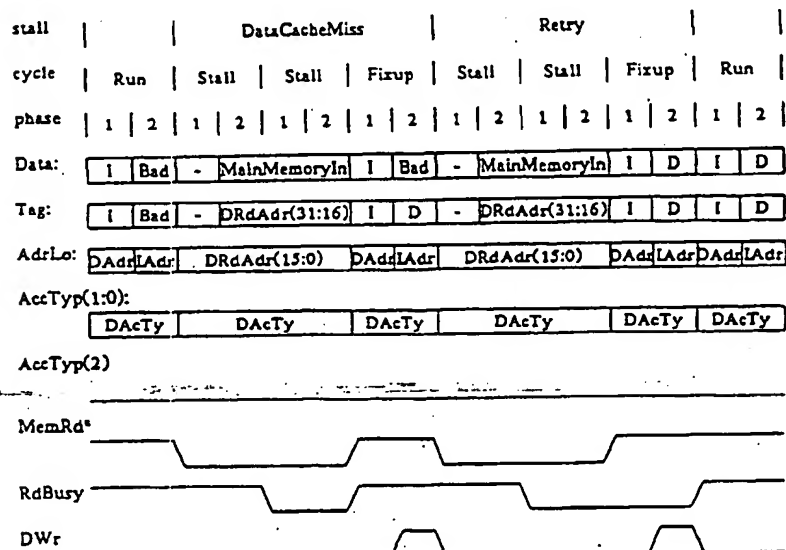
Figure 21: Data Cache Miss with Retry

The timing requirements for initiating a read or write busy retry are illustrated in figure 22.
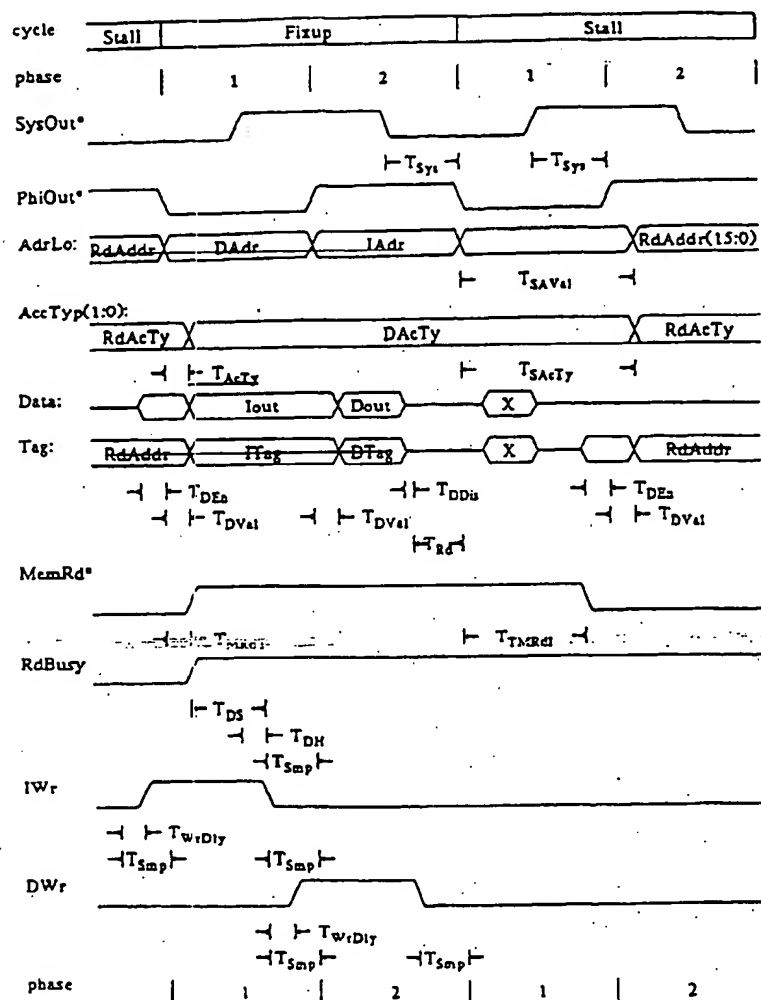
Figure 22: Read Busy Retry Timing

9. Interrupts

The processor has 6 general purpose interrupt inputs which are sampled during phase2 of all run and fixup cycles. After causing an interrupt exception to occur, the interrupts continue to be sampled during each phase2 to provide a level sensitive indication of the active interrupt or inputs. The interrupts are not latched within the processor when an interrrupt exception occurs. The timing of the interrupt inputs is illustrated in figure 23.
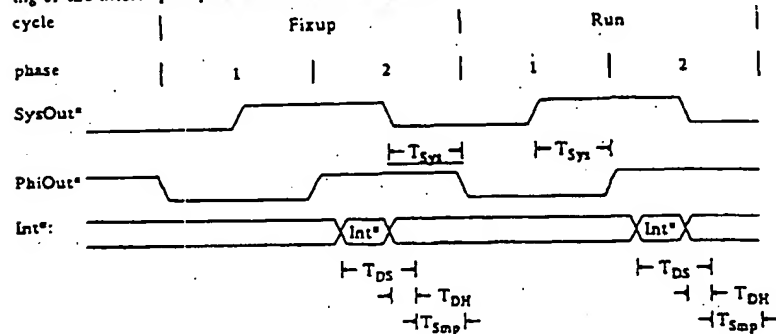
Figure 23: Interrupt Timing

The value of the interrupt inputs when Reset* is deasserted determine several processor operation modes. Each mode is described separately in the Advanced Features section.

10. Reset

The Reset* input is used to force processor execution starting at the reset exception vector and to initialize processor state. Reset* must be asserted for a minimum of six cycles to guarantee processor initialization. After a reset has occured the following processor state is guaranteed:

(1) KUc, the current Kernel/User bit, is zero corresponding to Kernel mode.

(2) IEc, the current interrupt enable bit, is zero corresponding to interrupts disabled.

(3) TS, the TLB shutdown bit, is zero corresponding to TLB enabled.

(4) SwC, the Swap Cache bit, is zero corresponding to caches not swapped.

(5) BEV, the Boot Exception Vector bit, is one corresponding to selection of the bootstrap exception vector.

(6) The Random register is set to zero.

When reset* is deasserted the processor latches the values present on the interrupt inputs. These values are used to determine various processor operating modes such as Endianness, Test, etc. A complete description of these modes is contained in the Advanced Features section. The operation of the processor coming out of reset is illustrated in figure 24.

stall										InstrCacheMiss		
cycle	Run	Run	Run	Run	Run	Stall	Fixup	Run
phase	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
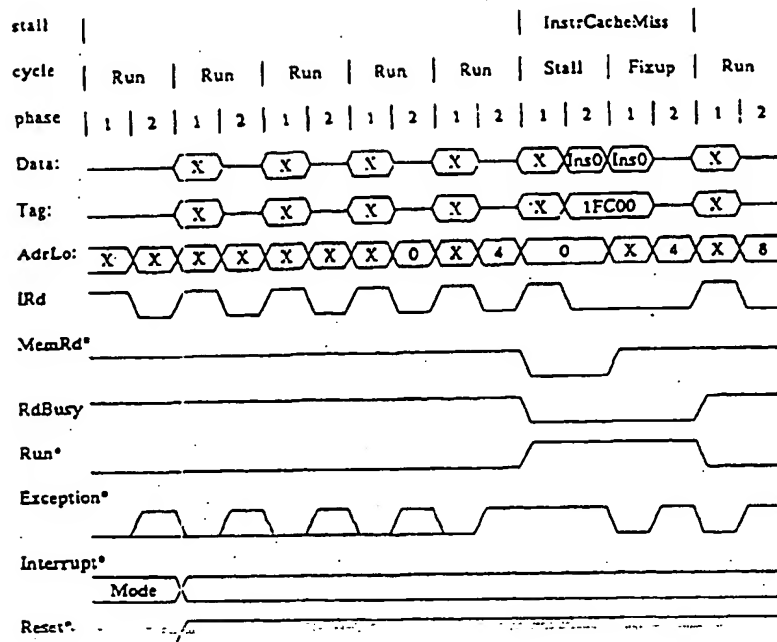
Figure 24: Reset Behavior

863 FH PG 0988

The timing requirements on reset* are illustrated in figure 25.
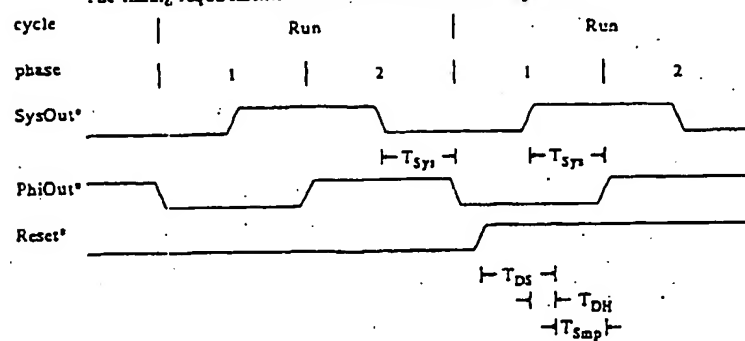
Figure 25: Reset Timing

863 FH PG 0989

11. Advanced Features

11.1. Cache Swapping

To facilitate cache flushing and diagnostics. the processor permits the instruction and data caches to be swapped. The cache which was acting as the instruction cache becomes the data cache and vice versa. Swapping the caches is accomplished by changing the value of the swap cache bit in the processor status register. All memory references which occur within the immediate vicinity of the swap must be uncached. Figure 26 illustrates the effects of cache swapping on the cache control signals. Although IWr and DWr are not shown. their relationships are reversed as well.

Figure 26: Cache Swapping

11.2. Cache Isolation

Cache diagnostics are further supported through a cache isolation capability. When the isolate cache bit of the processor status register is set. all loads hit in the cache and MemWr* is not asserted on stores.

11.3. Mode selectable features

The mode selectable features are determined by the state of the interrupt inputs during the reset period. Both phases of the clock are used for inputing mode information allowing for a total of 12 independently selectable features.

11.3.1. Phase 2 Modes

The following features are selected based upon the state of the interrupt inputs during phase 2 of the final cycle before reset* is deasserted.

11.3.1.1. Byte Order Control

Byte order or Endianness is determined by the value of Int*(0). Deassertion of Int*(0) will result in Little Endian ordering while assertion will result in Big Endian ordering.

11.3.1.2. Output Disable

Asserting Int*(1) causes the processor to tristate all of its outputs. In this condition the processor outputs can be driven by an external medium.

11.3.1.3. Cacheless Operation

The value of the interrupt Int*(2) determines whether caches are presumed present for instructions and data. Deassertion implies presence: assertion implies absence. When the caches are absent all memory references must occur at the processor cycle rate i.e.; no cache miss stalls can occur.

11.3.1.4. Data/Tag Drive Control

The value of the interrupt Int*(3) determines whether the data and tag buses are driven during write busy and coprocessor busy stalls. If asserted the buses are not driven during these stalls. When deasserted the data and tag buses are driven during phase 2 of the stall cycles. If the data and tag buses are not being driven externally during the aforementioned stalls, the processors drive should be enabled to prevent bus floating.

11.3.1.5. Phase Lock

Asserting the Int*(4) input causes the processor to insert additional phase delay into its input clock paths. The additional phase delay allows coprocessors to minimize their skew, i.e. *phase lock*, to the processor. Like the other mode inputs, the state of phase lock is latched at reset. However, if the phase locking mechanism is being used then the input must be asserted continuously for a period of time before the deassertion of reset so that the locking mechanism can stabilize. The phase lock time is determined by the slowest locking of the coprocessors. This feature is further described in the Coprocessor Interface section.

11.3.1.6. Output Timing Select

Deasserting Int*(5) selects the output timings based on the clock bindings described in this document. A summary of those clock bindings was presented in Section 3. Cache Timing.

11.3.2. Phase 1 Modes

The remaining features are selected based upon the state of the interrupt inputs during phase 1 of the final cycle before reset* is deasserted.

11.3.2.1. Phase 1 Mode Activate

Deasserting Int*(5) enables the remaining phase 1 modes. When Int*(5) is asserted all of the phase 1 modes default to their asserted conditions.

11.4. Mode Select Summary

The table below summarizes the processor's mode selectable features. Note that any activated reserved modes must be driven asserted to guarantee compatibility with future processor revisions.

Interrupt*	Phase 1		Phase 2	
	Asserted	Deasserted	Asserted	Deasserted
Int*(0)	Reserved	Reserved	Big Endian	Little Endian
Int*(1)	Reserved	Reserved	Tristate	Active
Int*(2)	Reserved	Reserved	Caches Absent	Caches Present
Int*(3)	Reserved	Reserved	Bus Drive On	Bus Drive Off
Int*(4)	Phase Delay On	Phase Delay Off	Phase Delay Off	Phase Delay On Off
Int*(5)	Phase 1 modes deactivated	Phase 1 modes activated	Rev 3 clock bindings	Rev 5 clock bindings

The timing requirements of the mode select inputs are illustrated in figure 27.
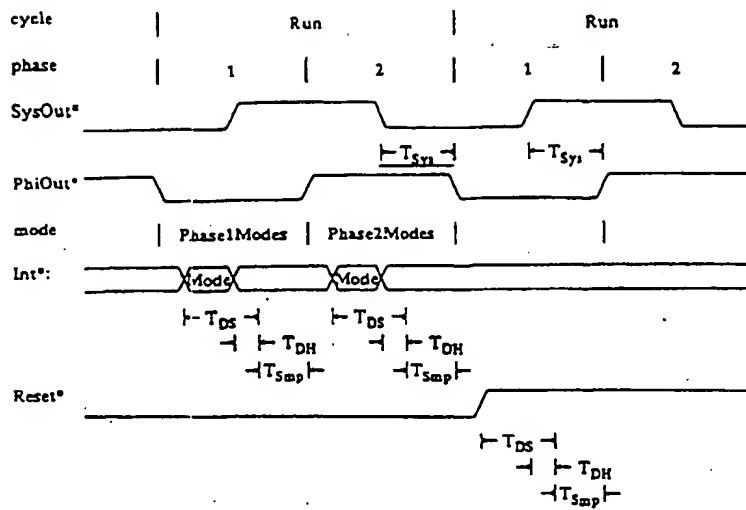
Figure 27: Mode Select Timing

12. Signal Summary

Data(31:0):
(i/o) A 32-bit bus used for all instruction and data transmission among the processor. caches. memory interface. and coprocessors.

DataP(3:0):
(i/o) A 4-bit bus containing even parity over the data bus.

Tag(31:12):
(i/o) A 20-bit bus used for transfering cache tags and high addresses between the processor. caches. and memory interface.

TagV:
(i/o) The tag validity indicator.

TagP(2:0):
(i/o) A 3-bit bus containing even parity over the catenation of TagV and Tag.

AdrLo(15:0):
(o) A 16-bit bus containing byte addresses used for transferring low addresses from the processor to the caches and memory interface.

IRd: (o) The read enable for the instruction cache.

IWr: (o) The write enable for the instruction cache.

IClk*:
(o) The instruction cache address latch clock. This clock runs continuously.

DRd: (o) The read enable for the data cache.

DWr:
(o) The write enable for the data cache.

DClk*:
(o) The data cache address latch clock. This clock runs continuously.

AccTyp(2:0):
(o) A 3-bit bus used to indicate the size of data being transferred on the data bus. whether or not a data transfer is occuring. and the purpose of the transfer.

MemWr*:
(o) Signals the occurence of a main memory write.

MemRd*:
(o) Signals the occurence of a main memory read.

BusError*:
(i) Signals the occurence of a bus error during a main memory read or write.

Run*:
(o) Indicates whether the processor is in the run or stall state.

Exception*:
(o) Indicates that the instruction about to commit state should be aborted.

SysOut*:
(o) A reflection of the internal processor clock used to generate the system clock.

CpSync*:
(o) A clock which is identical to SysOut* and used by coprocessors for timing synchronization with the CPU.

RdBusy:
(i) The main memory read stall termination signal.

WrBusy*:
(i) The main memory write stall initiation/termination signal.

CpBusy*:
    (i) The coprocessor busy stall initiation/termination signal.

CpCond(3:0):
    (i) A 4-bit bus used to transfer conditional branch status from the coprocessors to the main
    processor.

Int*(5:0):
    (i) A 6-bit bus used by the memory interface and coprocessors to signal maskable interrupts
    to the processor.

Clk2xSys:
    (i) The master double frequency input clock used for generating SysOut*.

Clk2xSmp:
    (i) A double frequency clock input used to determine the sample point for data coming into
    the processor and coprocessors.

Clk2xRd:
    (i) A double frequency clock input used to determine the enable time of the cache RAMs.

Clk2xPhi:
    (i) A double frequency clock input used to determine the position of the internal phases,
    phase1 and phase2.

Reset*:
    (i) Synchronous initialization input used to force execution starting from the reset memory
    address. Reset* must be synchronized by the leading edge of SysOut.

13. Pinout

Pin Name	Pin Number	Pin Name	Pin Number	Pin Name	Pin Number
Data(0)	E2	Tag(12)	B14	AdrLo(0)	C1
Data(1)	D1	Tag(13)	C13	AdrLo(1)	E3
Data(2)	F3	Tag(14)	D13	AdrLo(2)	D2
Data(3)	G2	Tag(15)	B15	AdrLo(3)	B1
Data(4)	G1	Tag(16)	E13	AdrLo(4)	C2
Data(5)	H2	Tag(17)	D14	AdrLo(5)	C4
Data(6)	H1	Tag(18)	C15	AdrLo(6)	A2
Data(7)	F2	Tag(19)	D15	AdrLo(7)	B3
Data(8)	H3	Tag(20)	E14	AdrLo(8)	C5
Data(9)	J3	Tag(21)	F14	AdrLo(9)	B4
Data(10)	J1	Tag(22)	G14	AdrLo(10)	A3
Data(11)	K2	Tag(23)	F15	AdrLo(11)	A4
Data(12)	L2	Tag(24)	H15	AdrLo(12)	B5
Data(13)	M1	Tag(25)	H14	AdrLo(13)	B7
Data(14)	N1	Tag(26)	J15	AdrLo(14)	A6
Data(15)	K1	Tag(27)	K15	AdrLo(15)	A7
Data(16)	M2	Tag(28)	J13	VCC0	F1
Data(17)	L3	Tag(29)	J14	VCC1	L1
Data(18)	N2	Tag(30)	L15	VCC2	Q1
Data(19)	N3	Tag(31)	L14	VCC3	N7
Data(20)	P2	TagP(0)	C14	VCC4	N8
Data(21)	Q2	TagP(1)	G15	VCC5	Q12
Data(22)	P4	TagP(2)	K14	VCC6	Q15
Data(23)	P1	TagV	N15	VCC7	M15
Data(24)	N5	Int*(0)	C9	VCC8	H13
Data(25)	Q3	Int*(1)	B9	VCC9	E15
Data(26)	P5	Int*(2)	A11	VCC10	A15
Data(27)	P6	Int*(3)	B10	VCC11	C8
Data(28)	Q5	Int*(4)	C10	VCC12	A5
Data(29)	Q7	Int*(5)	A12	VCC13	C3
Data(30)	P8	CpCond(0)	A8	VCC14	A1
Data(31)	Q4	CpCond(1)	B8	Gnd0	D3
DataP(0)	E1	CpCond(2)	A9	Gnd1	G3
DataP(1)	J2	CpCond(3)	A10	Gnd2	K3
DataP(2)	M3	AccTyp(0)	P15	Gnd3	N4
DataP(3)	N6	AccTyp(1)	M14	Gnd4	Q6
Clk2xSys	P9	AccTyp(2)	L13	Gnd5	N9
Clk2xSmp	Q10	MemWr*	N12	Gnd6	N10
Clk2xRd	P10	MemRd*	N13	Gnd7	M13
Clk2xPhi	Q9	Run*	N14	Gnd8	K13
RdBusy	C11	IRd	P12	Gnd9	G13
WrBusy*	A13	IWr	P13	Gnd10	F13
CpBusy*	B11	DRd	N11	Gnd11	C12
BusError*	B12	DWr	Q14	Gnd12	C7
Reset*	A14	IClk*	Q13	Gnd13	C6
SysOut*	Q11	DClk*	P11	Exc*	Q8
CpSync*	P14	Resvd0	P3	Resvd1	P7
Resvd2	B2	Resvd3	B6	Resvd4	B13

Table I: Pinout - 144 pin PGA

Physical Pin Placement

QED 00053
CONFIDENTIAL

Pinout

863 FH PG 0997

14. Timing Parameters

14.1. DC Characteristics

14.1.1. Maximum Ratings (Operation beyond the limits set forth in this table may impair the useful life of the device.)

Parameter	Symbol	Test Conditions	Min	Max	Units
Supply Voltage	VCC		-.5	+7.0	V
Input Voltage	VIN		-.5[1]	+7.0	V
Storage Temperature	TST		-65	+150	C
Operating Temperature	TA		0	+70	C
Load Capacitance on any Pin	CLd			100	pF

Note:

(1) VIN Min. = -3.0V for pulse width less than 15ns.

(2) Not more than one output should be shorted at a time. Duration of the short should not exceed 30 seconds.

14.1.2. Operating Range

Range	Ambient Temperature	VCC
Commercial	0C to 70C	5V ± 5%

14.1.3. Operating Parameters

Parameter	Symbol	Conditions	12.5 MHz		16.67 MHz		
			Min	Max	Min	Max	Units
Output HIGH Voltage	VOH	VCC = Min. IOH = -4mA	3.5		3.5		V
Output LOW Voltage	VOL	VCC = Min. IOL = 4mA		.4		.4	V
Input HIGH Voltage	VIH		2	VCC+.5	2	VCC+.5	V
Input LOW Voltage	VIL		-.5[1]	.8	-.5[1]	.8	V
Input HIGH Voltage	VIHS		2.5	VCC+.5	3.0	VCC+.5	V
Input LOW Voltage	VILS		-.5[1]	.4	-.5[1]	.4	V
Input Capacitance	CIn		10		10		pF
Output Capacitance	COut		10		10		pF
Operating Current	ICC	VCC = 5.5V		250		300	mA

Note:

(1) VIL Min. = -3.0V for pulse width less than 15ns.

(2) VIHS and VILS apply to Clk2xSys, Clk2xSmp, Clk2xRd, Clk2xPhi, CpBusy, and Reset*.

14.2. AC Characteristics

Notes:

(1) All output timings are given assuming 25pf of capacitive load. Output timings should be derated where appropriate as per the table below.

(2) All timings referenced to 1.5V

14.2.1. Clock Parameters

Parameter	Symbol	Test Conditions	12.5 MHz		16.67 MHz		
			Min	Max	Min	Max	Units
Input Clock High	TCkHigh	Transition ≤5ns	18		13.5		ns
Input Clock Low	TCkLow	Transition ≤5ns	18		13.5		ns
Input Clock Period	TCkP		40	1000	30	1000	ns
Clk2xSys to Clk2xSmp			0	$\frac{t_{cyc}}{4}$	0	$\frac{t_{cyc}}{4}$	ns
Clk2xSmp to Clk2xRd			0	$\frac{t_{cyc}}{4}$	0	$\frac{t_{cyc}}{4}$	ns
Clk2xSmp to Clk2xPhi			11	$\frac{t_{cyc}}{4}$	9	$\frac{t_{cyc}}{4}$	ns

Note:

(1) The clock parameters apply to all four 2xClocks: Clk2xSys, Clk2xSmp, Clk2xRd, and Clk2xPhi.

14.2.2. Run Operation Parameters

Parameter	Load (pf)	Symbol	12.5 MHz		16.67 MHz	
			Min (nsec)	Max (nsec)	Min (nsec)	Max (nsec)
Data Enable		T_{DEn}	-1	-2.5	-1	-2
Data Disable		T_{DDis}	0	-1	0	-1
Data Valid	25	T_{DV}	2	3.5	2	3
Write Delay	25	T_{WrDly}	0	7.5	0	5
Data Setup		T_{DS}	11.5		9	
Data Hold		T_{DH}	-4		-4	
CpBusy Setup		T_{CBS}	15		13	
CpBusy Hold		T_{CBH}	-4		-4	
Access Type(1:0)	25	T_{AcTy}	1	10	1	7
Access Type(2)	25	T_{AcTy2}	1	20	1	17
Memory Write	25	T_{MWr}	1	10	1	7
Exception	25	T_{Exc}	1	10	1	7

863 FH PG 0999

14.2.3. Stall Operation Parameters

Parameter	Load (pf)	Symbol	12.5 MHz		16.67 MHz	
			Min (nsec)	Max (nsec)	Min (nsec)	Max (nsec)
Address Valid	25	T_{LAVal}		38		30
Access Type	25	T_{SAcTy}		35		27
Memory Read Initiate	25	T_{MRsI}	1	35	1	27
Memory Read Terminate	25	T_{MRsT}	1	10	1	7
Run Terminate	25	T_{St}	5	25	5	17
Run Initiate	25	T_{Run}	5	15	5	12
Memory Write	25	T_{SMWr}	5	35	5	27
Exception Valid	25	T_{STxc}	5	28	5	20

14.2.4. Capacitive Load Deration

Parameter	Symbol	Conditions	12.5 MHz		16.67 MHz		
			Min	Max	Min	Max	Units
Load Derate	CLD		1	2.5	1	2	nsec/25pF

863 FH PG 1000

15. Cache Design

Figure 28 illustrates the timing critical portions of an R2000 system which has a 64 KByte instruction and a 64 KByte data cache. The caches are built out of 16Kx4 static RAMS.

The tables below contain the timing parameters used in the cache design for the static RAMS and the TTL logic, respectively.

Parameter	Load (pf)	Symbol	12.5 MHz		16.67 MHz	
			Min (nsec)	Max (nsec)	Min (nsec)	Max (nsec)
Address to Data Valid	30	T_{AA}		35		25
Output Enable to Data Valid	30	T_{DOE}		20		15
Output Disable Time		T_{HZ}	2.5	15	2	10
Output Enable Time		T_{LZ}	2.5		2	
Address Setup to End of Write		T_{AW}		30		20
Data Setup to End of Write		T_{SD}	15		13	
Write Pulse Width		T_{PWE}	30		20	
Data Hold from End of Write		T_{HD}	0		0	
Address Hold from End of Write		T_{HA}	0		0	
Enable/Disable Mismatch		T_{EDM}		6		2

Cache Ram Parameters

Note:

(1) T_{HZ} is computed as TDOE*2/3

(2) T_{LZ} is computed as TDOE/8

(3) T_{EDM} assumes that when operating at approximately the same voltage and temperature that the enable and disable times of the highest speed grade cache RAMS will match to within 20%. This assures a 10% variation in gate length at the highest grade and a square law dependence of speed with gate length. To allow for down binning; that is, selling a higher speed part to a lower speed specification, a 40% mismatch is assumed for the second fastest speed grade.

Parameter	Load (pf)	Symbol	Min (nsec)	Typ (nsec)	Max (nsec)
F373 Propagation Delay	50	$T_{373_{PD}}$			8
F373 Latch Enable Delay	50	$T_{373_{LE}}$	3		13
F373 Latch Enable Hold	50	$T_{373_{Hd}}$	3		13
1804 NAND Buffer	50	T_{1804}	1		4
F241 Disable	50	T_{241}	2		7

Cache Logic Parameters

Notes:

(1) The TTL and cache RAM propagation delays are derated by 1 nsec per 25pf of additional load.

(2) Cache RAM input capacitance is 5pf

(3) Cache RAM output capacitance is 7pf

(4) The outputs of an 1804 TTL NAND buffer are assumed to match to within one nsec.

(5) SysClk is a buffered version of SysOut*

(6) This analysis assumes a system which has sufficiently well controlled Data and Tag bus characteristics to guarantee approximately 5 nsec of hold time on these buses. The easiest way to guarantee these conditions is by using only MOS devices on the Data and Tag buses.
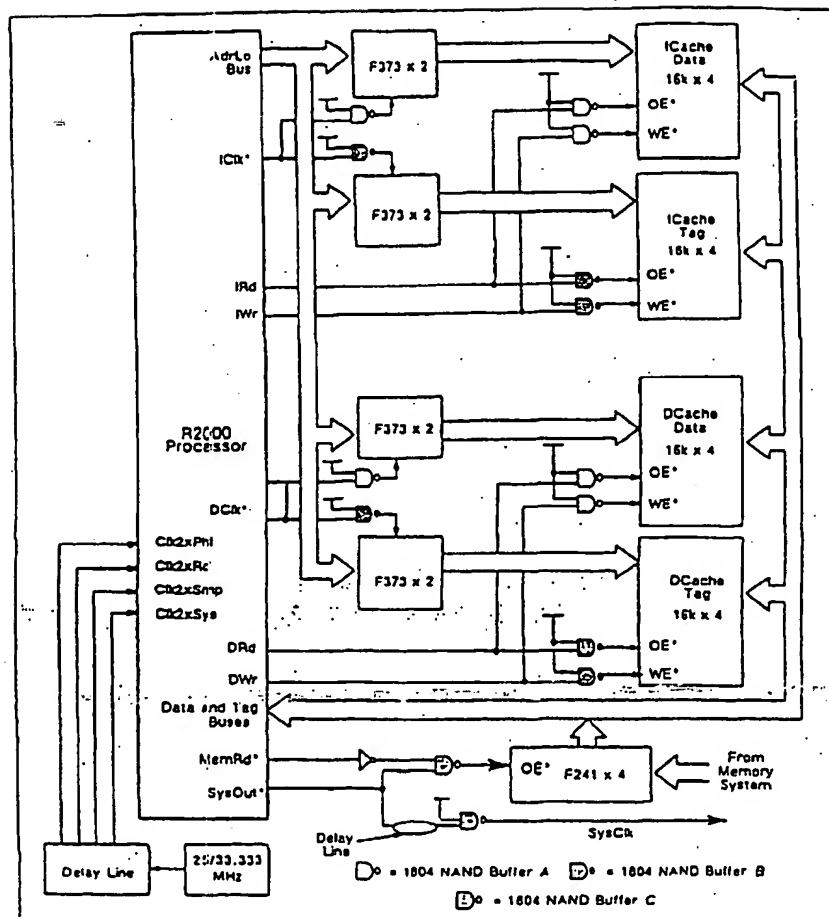
Figure 28. 64 Kbyte Instruction/Data Cache Configuration

[Handwritten note at right: These don't go to the DRAM chip, but to the interface]

- 54 -

15.1. Cache Design Overview   Design of the cache subsystem is principally a matter of placing the four 2x input clocks so as to maximize performance for a given set of static RAM parameters. Recall that the relationships of the four 2x input clocks are defined as shown below.

Clk2xPhi

Clk2xRd

Clk2xSmp

Clk2xSys

$$\dashv \vdash T_{Rd}$$
$$\dashv T_{Smp} \vdash$$
$$\vdash T_{Sys} \dashv$$
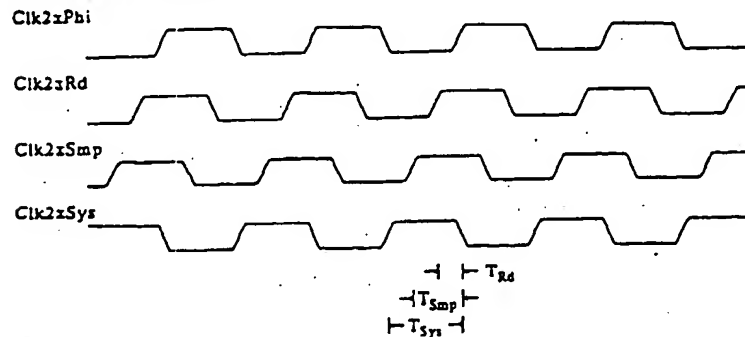
2x Input Clocks

The primary functions of each of the 2xClocks with regard to the cache subsystem are as follows:

(1) Clk2xSys determines the position of SysOut*. Clk2xSys is positioned to prevent ICache to DCache contention on back to back reads and system bus to cache bus contention at the end of read stalls.

(2) Clk2xSmp determines the sample point for data coming into the processor, terminates cache write strobes, and terminates the address latch clocks, IClk* and DClk*. Clk2xSmp is positioned to guarantee sufficient propagation time through the processor load aligner.

(3) Clk2xRd initiates the cache read strobes, terminates the drive of the data and tag buses, and initiates the address latch clocks. Clk2xRd is positioned to guarantee sufficient data setup to sample.

(4) Clk2xPhi initiates the drive of the major processor outputs: address, data, and tag. Clk2xSys. Clk2xSmp, and Clk2xRd are positioned relative to Clk2xPhi.

The principal equations governing the placement of TSmp and TRd for cache reads and cache writes are illustrated in figures 29a and 29b, respectively.

863 FH PG 1004

instr | Load |

phase | 1 | 2 |

SysOut*

$\vdash\ T_{Sys}\ \dashv$

PhiOut*

AdrLo: | IAdr | DAdr | IAdr | DAdr

$\dashv\ \vdash T_{AdrLo_{Derate}}$

$\vdash_{373_{PD}}\dashv$

$\dashv\vdash\ T_{373_{PD_{Derate}}}$

$\vdash\quad T_{RAM_{AA}}$

$\dashv\ \vdash\ T_{RAM_{AA_{Derate}}}$

$\vdash T_{DS}\dashv$

$\vdash T_{Smp}\dashv$

T_{Cyc}

Address access to sample

$$t_{Smp}\ \le\ t_{Cyc}\ -\ (t_{AdrLo_{Derate}}\ +\ t_{373_{LN}}\ +\ t_{373_{PD_{Derate}}}\ +\ t_{RAM_{AA}}\ +\ t_{RAM_{AA_{Derate}}}\ +\ t_{DS})\tag{3}$$

DRd

$\vdash t_{Rd}\dashv$

$\dashv\ \vdash\ T_{1804_{Max}}$

$\vdash T_{RAM_{DOE}}\dashv$

$\dashv\ \vdash\ T_{RAM_{DOE_{Derate}}}$

$\vdash T_{DS}\dashv$

$\vdash T_{Smp}\dashv$

$\vdash\quad T_{Cyc/2}\quad\dashv$

Data: ─────────────────────⟨Din⟩─────────────

Output enable to sample

$$t_{Smp-Rd}\ \le\ t_{\frac{Cyc}{2}}\ -\ (t_{1804_{Max}}\ +\ t_{RAM_{DOE}}\ +\ t_{RAM_{DOE_{Derate}}}\ +\ t_{DS})\tag{8}$$
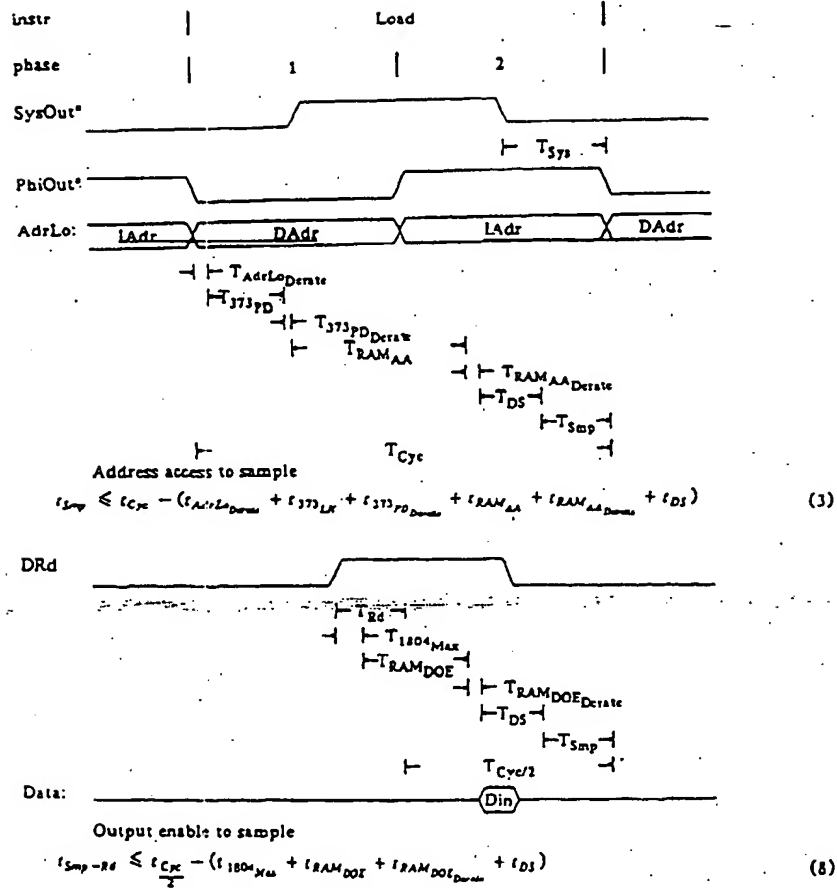
Figure 29a: Primary cache read timing

Figure 29b: Primary cache write timing

The timing diagram shows signals: instr / phase (1, Store, 2), SysOut*, PhiOut*, AdrLo (IAdr, DAdr, IAdr, DAdr), Data (Dout), DWr.
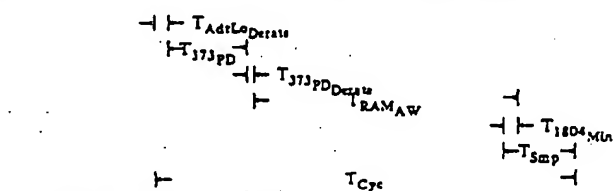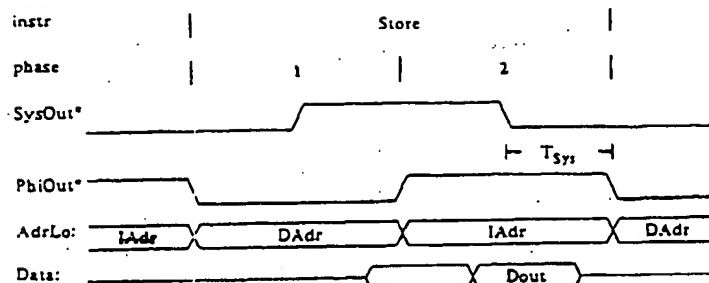
Address setup to end of write

$$t_{Smp} \leq t_{Cyc} - (t_{AdrLo_{Derate}} + t_{373_{LH}} + t_{373PD_{Derate}} + t_{RAM_{AW}}) + t_{1804_{Min}} \tag{4}$$

Data setup to end of write

$$t_{Smp} \leq t_{\frac{Cyc}{2}} - t_{DVal} - t_{DVal_{Derate}} - t_{RAM_{SD}} + t_{1804_{Min}} \tag{5}$$

15.1.1. Operation Constraints

The following pages present the equations for determining the positions of the 2x input clocks. The assumed loading on the address and data bus is 50pF and 75pF respectively.

15.1.1.1. t_{Smp} constraints

Internal Sample to Phase delay

$$t_{Smp} \geq t_{Smp_{Min}} \tag{1}$$

12.5 MHz
$$t_{Smp} \geq 11$$
16.67 MHz
$$t_{Smp} \geq 9$$

Address capture by IClk* and DClk*

pbase

SysOut*

PhiOut*

AdrLo: | IAdr | DAdr | IAdr | DAdr |

IClk*

$T_{1804_{Max}}$
$T_{373_{Hld}}$
$T_{AdrLo_{Derate_{Min}}}$
T_{Smp}

DClk*

$T_{1804_{Max}}$
$T_{373_{Hld}}$
$T_{AdrLo_{Derate_{Min}}}$
T_{Smp}
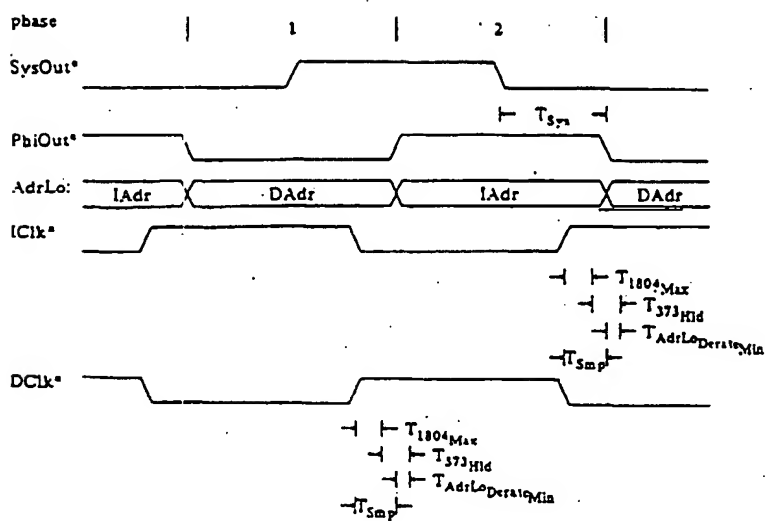
$$t_{Smp} \geq t_{1804_{Max}} + t_{373_{Hld}} - t_{AdrLo_{Derate_{Min}}}$$ (2)

12.5 MHz

$\geq 4 + 3 - 1$

≥ 6

16.67 MHz

$\geq 4 + 3 - 1$

≥ 6

863 FH PG 1008

Address access to sample - assuming address delay through F373 is limited by its propagation delay rather than its clock delay
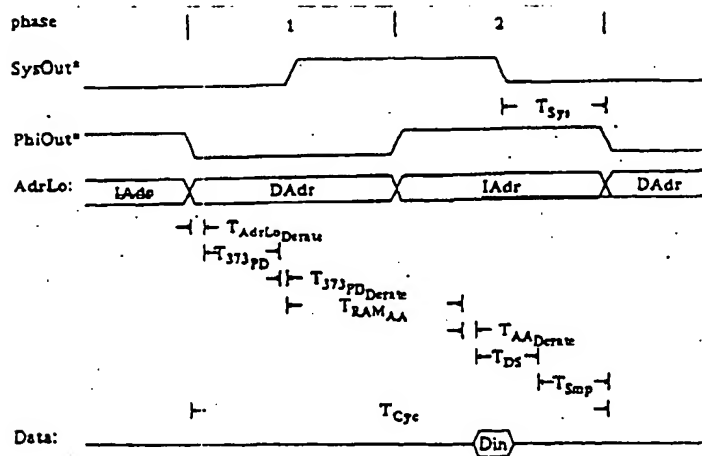
$$t_{Smp} \leq t_{Cyc} - (t_{AdrLo_{Derate}} + t_{373_{PD}} + t_{373_{PD_{Derate}}} + t_{RAM_{AA}} + t_{RAM_{AA_{Derate}}} + t_{DS}) \qquad (3)$$

12.5 MHz

$$\leq 80 - (2.5 + 8 + 1 + 35 + 2 + 11.5)$$

$$\leq 20$$

16.67 MHz

$$\leq 60 - (2 + 8 + 1 + 25 + 2 + 9)$$

$$\leq 13$$

863 FH PG 1009

Address setup to end of write
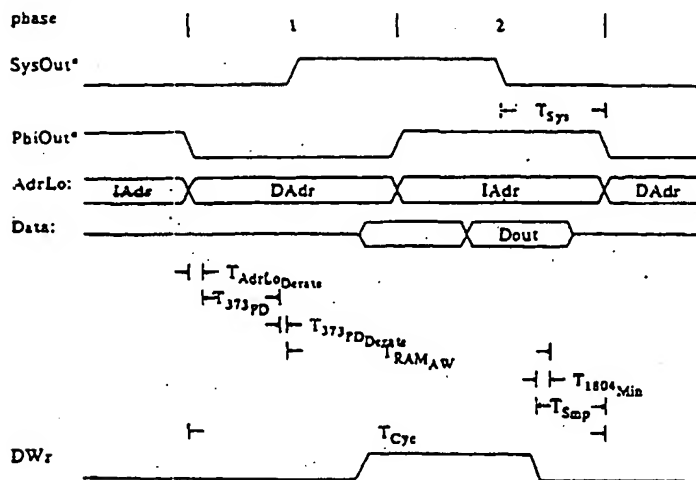
$$t_{Smp} \leqslant t_{Cyc} - (t_{AdrLo_{Derate}} + t_{373_{LH}} + t_{373_{PD_{Derate}}} + t_{RAM_{AW}}) + t_{1804_{Min}} \tag{4}$$

12.5 MHz

$$\leqslant 80 - (25 + 8 + 1 + 30) + 1$$

$$\leqslant 39.5$$

16.67 MHz

$$\leqslant 60 - (2 + 8 + 1 + 20) + 1$$

$$\leqslant 30$$

Cache Design

Data setup to end of write
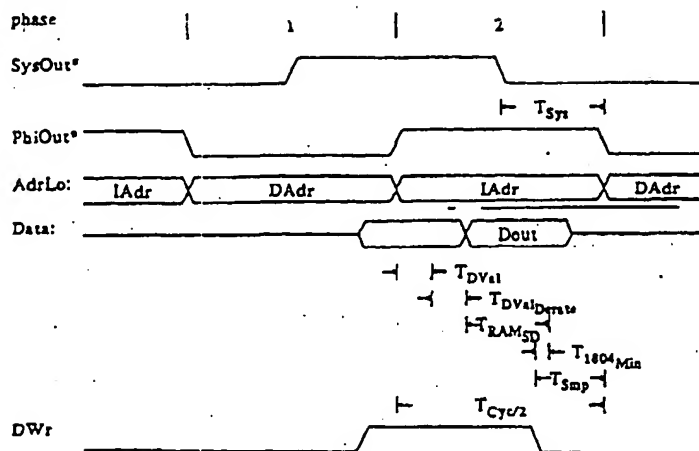
$$t_{Smp} \leq t_{\frac{Cyc}{2}} - t_{DVal} - t_{DVal_{Dwrite}} - t_{RAM_{SD}} + t_{1804_{Min}}$$ (5)

12.5 MHz

$$\leq 40 - 3.5 - 5 - 15 + 1$$

$$\leq 17.5$$

16.67 MHz

$$\leq 30 - 3 - 4 - 13 + 1$$

$$\leq 11$$

. 863 FH PG 1011

15.1.1.2 t_{Rd} constraints

Guarantee minimum delay through transparent latches
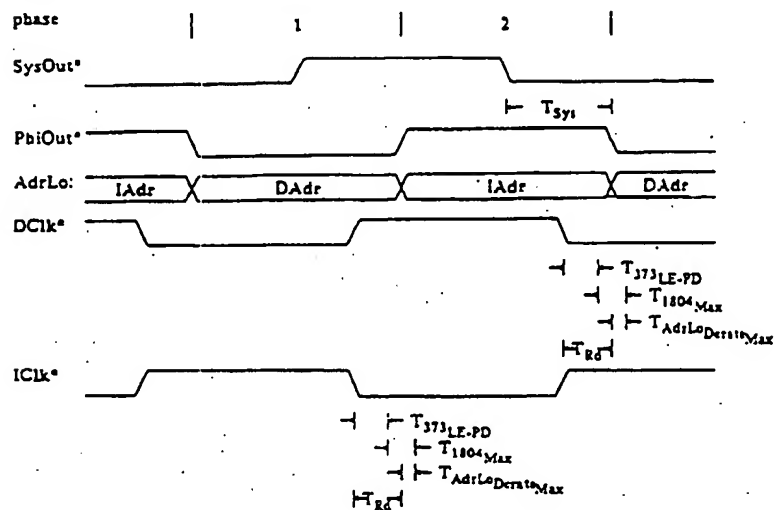
$$t_{Rd} \geq (t_{373_{LE}} - t_{373_{PD}}) + t_{1804_{Max}} - t_{AdrLo_{Derate_{Max}}} \tag{6}$$

12.5 MHz

$\geq 5 + 4 - 2.5$

≥ 6.5

16.67 MHz

$\geq 5 + 4 - 2$

≥ 7

Cache Design

863 FH PG 1012

15.1.1.3. t_{Syt-Rd} constraints

Minimum read pulse width
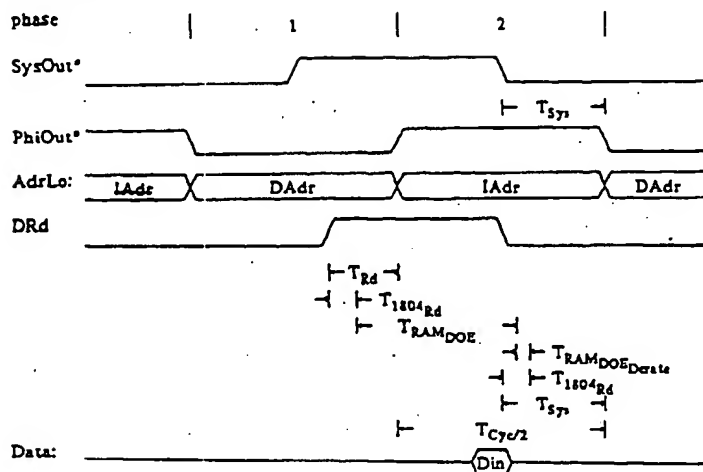
$$t_{Syt-Rd} \leq t_{\frac{Cyc}{2}} - (t_{1804_{Rd}} + t_{RAM_{DOE}} + t_{RAM_{DOE_{Derate}}}) + t_{1804_{Rd}} \tag{7}$$

12.5 MHz

$$\leq 40 - (20 + 2)$$

$$\leq 18$$

16.67 MHz

$$\leq 30 - (15 + 2)$$

$$\leq 13$$

15.1.1.4. t_{Smp-Rd} constraints

Output enable to sample
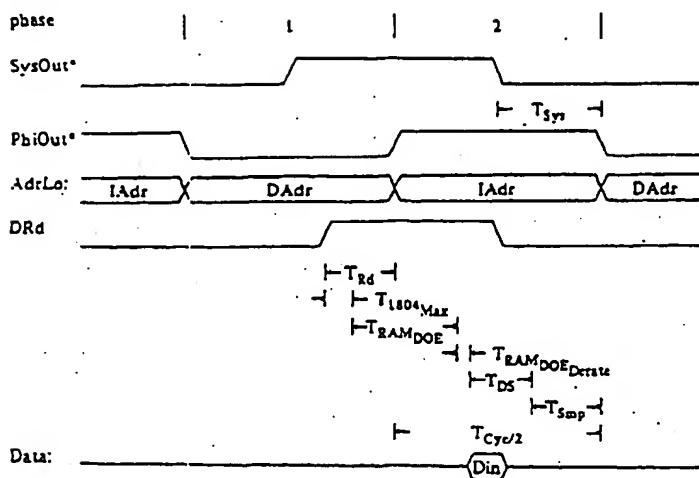
$$t_{Smp-Rd} \leq \frac{t_{Cyc}}{2} - (t_{1804_{Max}} + t_{RAM_{DOE}} + t_{RAM_{DOE_{Derate}}} + t_{DS}) \qquad (8)$$

12.5 MHz

$$\leq 40 - (4 + 20 + 2 + 11.5)$$

$$\leq 2.5$$

16.67 MHz

$$\leq 30 - (4 + 15 + 2 + 9)$$

$$\leq 0$$

Data hold from end of write - Assuming write data holds on bus until subsequent read
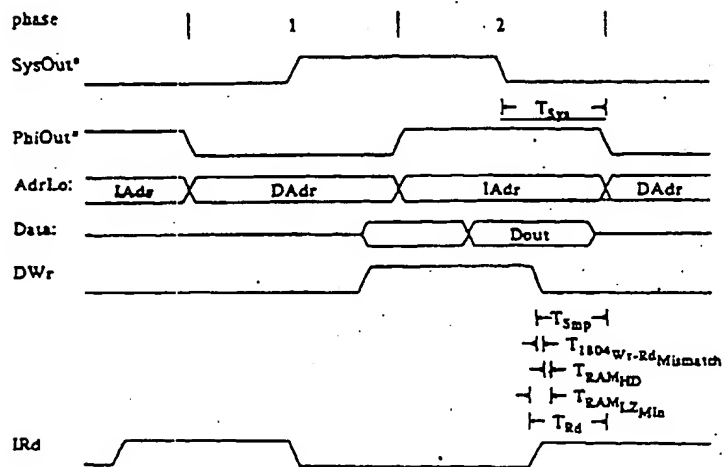
$$t_{Smp-Rd} \geq t_{1804_{Wr-Rd_{Mismatch}}} + t_{RAM_{HD}} - t_{RAM_{LZ_{Min}}} \quad \ldots (3)$$

12.5 MHz

$$\geq 1 + 0 - 2$$

$$\geq (-1)$$

16.67 MHz

$$\geq 1 + 0 - 2$$

$$\geq (-1)$$

Cache Design

Address hold from end of write

phase
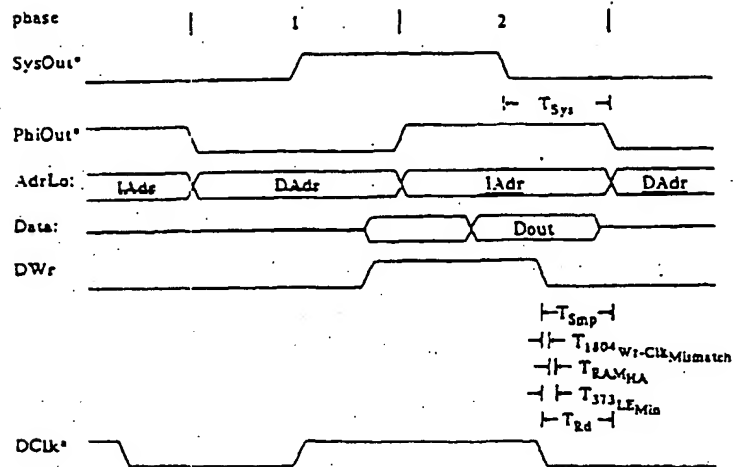
$$t_{Smp-Rd} \geq t_{1804_{Wr-Clk_{Mismatch}}} - t_{373_{LE_{Min}}} + t_{RAM_{HA}} \tag{10}$$

12.5 MHz

$$\geq 1 - 3 + 0$$

$$\geq -2$$

16.67 MHz

$$\geq 1 - 3 + 0$$

$$\geq -2$$

15.1.1.5. $t_{C_{pc}}$ constraints

Minimum write pulse width
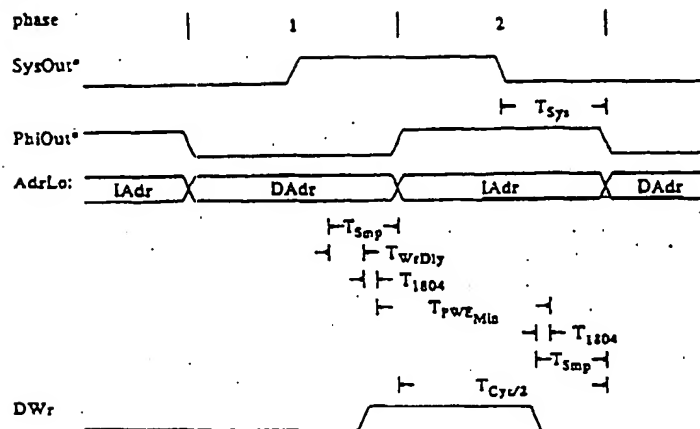
$$\frac{t_{C_{pc}}}{2} \geq t_{Smp} + t_{WrDly} + t_{1804} + t_{PWZ_{min}} - t_{1804} - t_{Smp}$$

(II)

12.5 MHz

$$\geq 7.5 + 30$$

$$\geq 37.5$$

16.67 MHz

$$\geq 5 + 20$$

$$\geq 25$$

15.1.1.6. $t_{DBu_{Rd}}$ constraints

Cache output valid at Smp
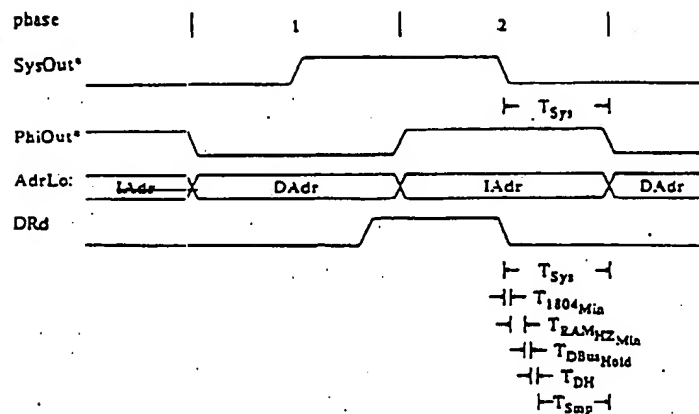
$$t_{DBu_{Hold}} \geq t_{Sys-Smp} - (t_{1804_{Min}} + t_{RAM_{HZ_{Min}}} - t_{DH}) \qquad\qquad (12)$$

12.5 MHz

$$\geq t_{Sys-Smp} - (1 + 2 - (-4))$$

$$\geq t_{Sys-Smp} - 7$$

16.67 MHz

$$\geq t_{Sys-Smp} - (1 + 2 - (-4))$$

$$\geq t_{Sys-Smp} - 7$$

863 FH PG 1018

Data valid at end of write
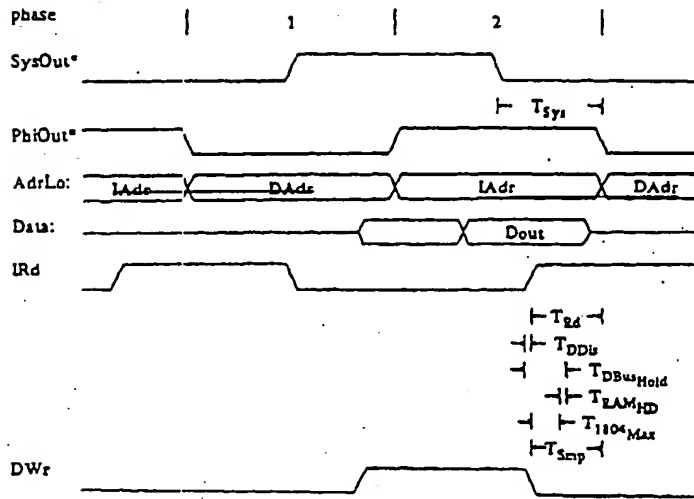
phase

$$t_{DBw_{Max}} \geq t_{1804_{Max}} - t_{DDw_{Max}} + t_{RAM_{HD}} - t_{Smp-Rd} \qquad (13)$$

12.5 MHz

$$\geq 4 - (-1) - t_{Smp-Rd}$$

$$\geq 5 - t_{Smp-Rd}$$

16.67 MHz

$$\geq 4 - (-1) - t_{Smp-Rd}$$

$$\geq 5 - t_{Smp-Rd}$$

863 FH PG 1019

Data hold from main memory read
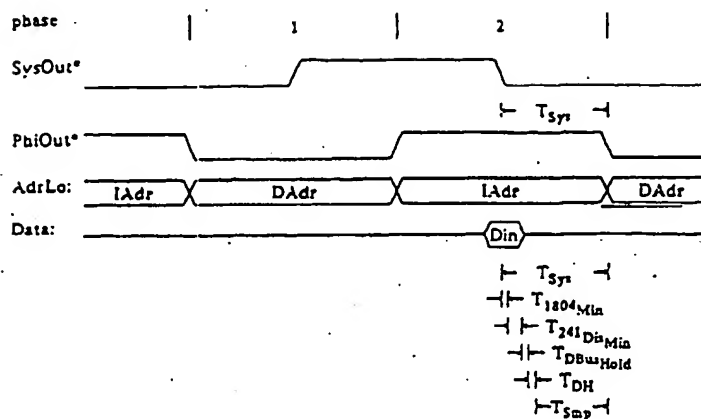
$$t_{DBus_{Hold}} \geq t_{Sys-Smp} - (t_{1804_{Min}} + t_{241_{Din_{Min}}} - t_{DH}) \qquad (14)$$

12.5 MHz

$$\geq t_{Sys-Smp} - (1 + 2 - (-4))$$

$$\geq t_{Sys-Smp} - 7$$

16.67 MHz

$$\leq t_{Sys-Smp} - (1 + 2 - (-4))$$

$$\leq t_{Sys-Smp} - 7$$

15.1.2. Contention Constraints

15.1.2.1. t_{Sys-Rd} constraints

Read - Read. ICache - DCache contention
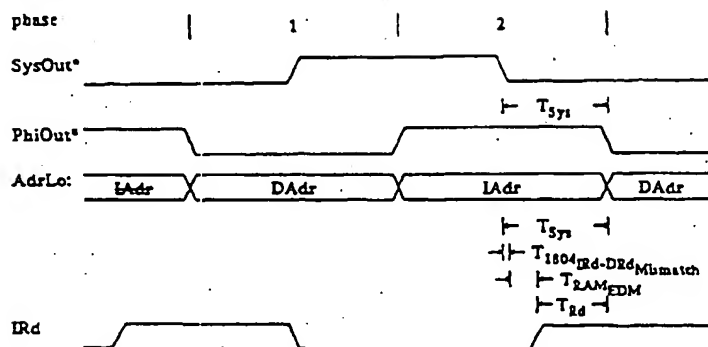
$$t_{Sys-Rd} \geq t_{1804_{IRd-DRd_{Mismatch}}} + t_{RAM_{EDM}} \tag{15}$$

12.5 MHz

$\geq 1 + 6$

≥ 7

16.67 MHz

$\geq 1 + 2$

≥ 3

15.1.2.2. $t_{S_{yr}}$ constraints

Read - Write. ICache - Data Bus contention

$$t_{S_{yr}} \geq t_{1804_{Max}} + t_{RAM_{HZ}} - t_{DE_n} \qquad\qquad (16)$$

12.5 MHz

$\geq 4 + 15 - (-2.5)$

≥ 21.5

16.67 MHz

$\geq 4 + 10 - (-2)$

≥ 16
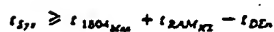
Main Memory - Data Bus contention - end of stall
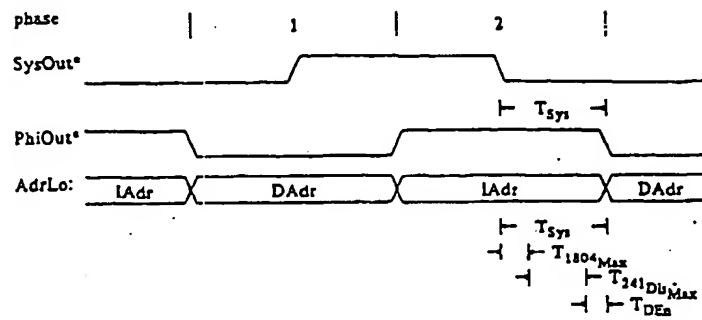
$$t_{Sys} \geq t_{1804_{Max}} + t_{241_{Du_{Max}}} - t_{DEn} \qquad\qquad (17)$$

12.5 MHz

$$\geq 4 + 7 - (-2.5)$$

$$\geq 13.5$$

16.67 MHz

$$\geq 4 + 7 - (-2)$$

$$\geq 13$$

15.1.3. System Design Constraints

SysClk delay required to guarantee data input valid at processor sample point - Assuming no clock to output delay on register

phase

SysOut*

PhiOut*

AdrLo:
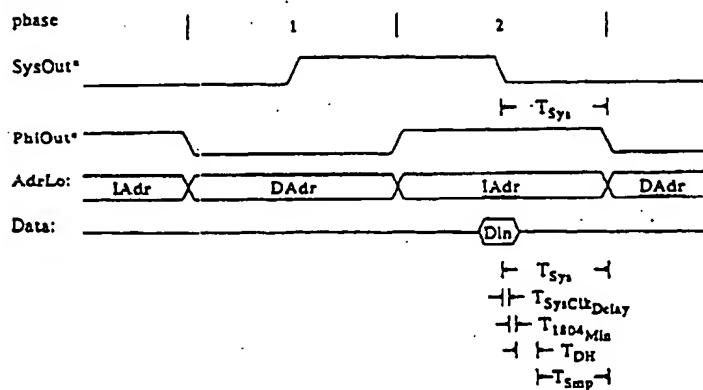
Data:

$$t_{SysClk_{Delay}} \geq t_{Sys-Smp} - t_{1804_{Min}} + t_{DH} \tag{18}$$

12.5 MHz

$$\geq t_{Sys-Smp} - 1 + (-4)$$

$$\geq t_{Sys-Smp} - 5$$

16.67 MHz

$$\geq t_{Sys-Smp} - 1 + (-4)$$

$$\geq t_{Sys-Smp} - 5$$

Data setup to SysClk - $t_{Setup_{SysClk}}$
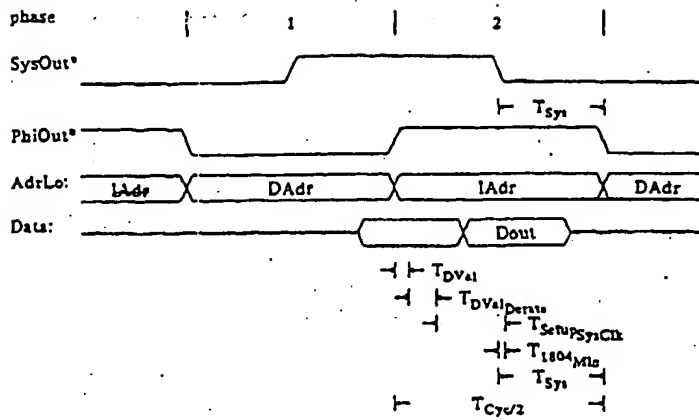
$$t_{Setup_{SysClk}} = \frac{t_{Cyc}}{2} - t_{DVal} - t_{DVal_{Derate}} - t_{Sys} + t_{1804_{Min}}$$ (19)

12.5 MHz

$$= 40 - 3.5 - 5 - t_{Sys} + 1$$

$$= 32.5 - t_{Sys}$$

16.67 MHz

$$= 30 - 3 - 4 - t_{Sys} + 1$$

$$= 24 - t_{Sys}$$

Data hold from SysClk $t_{Hold_{SysClk}}$ - Assuming data holds on bus until subsequent read

phase --

SysOut*

$\vdash T_{Sys} \dashv$

PhiOut*

AdrLo: IAdr   DAdr   IAdr   DAdr

Data: Dout

$\vdash T_{Sys} \dashv$
$\dashv \vdash T_{1804_{Sys_{Max}}}$
$\dashv \vdash T_{Hold_{SysClk}}$
$\dashv \vdash T_{RAM_{L2_{Min}}}$
$\dashv \vdash T_{1804_{Rd_{Min}}}$
$\vdash T_{Rd} \dashv$

IRd
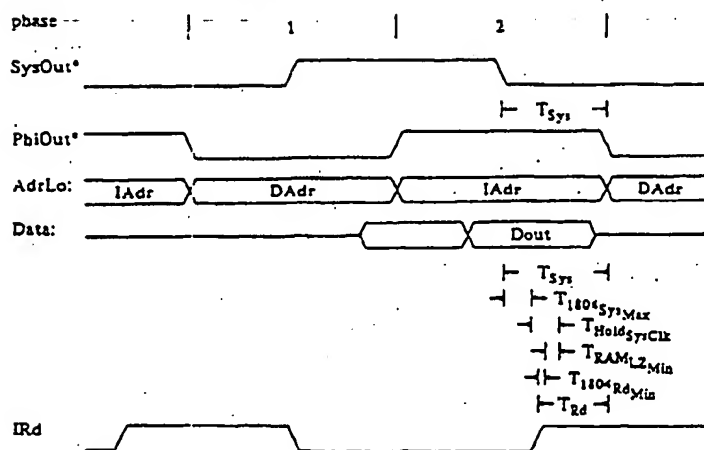
$$t_{Hold_{SysClk}} = t_{Sys-Rd} - t_{1804_{Sys_{Max}}} + t_{RAM_{L2_{Min}}} + t_{1804_{Rd_{Min}}} \tag{20}$$

12.5 MHz

$$= t_{Sys-Rd} - 4 + 2 + 1$$

$$= t_{Sys-Rd} - 1$$

16.67 MHz

$$= t_{Sys-Rd} - 4 + 2 + 1$$

$$= t_{Sys-Rd} - 1$$

863.FH PG 1026

15.1.4. Summary

15.1.4.1. Operation Constraints

12.5MHz

$$11 \leq t_{Smp} \leq 17.5 \qquad\qquad\qquad (1.5)$$

$$t_{Rd} \geq 6.5 \qquad\qquad\qquad (6)$$

$$t_{Srt-Rd} \leq 18 \qquad\qquad\qquad (7)$$

$$-1 \leq t_{Smp-Rd} \leq 2.5 \qquad\qquad\qquad (8.9)$$

$$\frac{t_{Cyc}}{2} \geq 37.5 \qquad\qquad\qquad (11)$$

16.67 MHz

$$9 \leq t_{Smp} \leq 11 \qquad\qquad\qquad (1.5)$$

$$t_{Rd} \geq 7 \qquad\qquad\qquad (6)$$

$$t_{Srt-Rd} \leq 13 \qquad\qquad\qquad (7)$$

$$-1 \leq t_{Smp-Rd} \leq 0 \qquad\qquad\qquad (8.9)$$

$$\frac{t_{Cyc}}{2} \geq 25 \qquad\qquad\qquad (11)$$

15.1.4.2. Contention Constraints

12.5 MHz

$$t_{Srt-Rd} \geq 7 \tag{15}$$

$$t_{Srt} \geq 21.5 \tag{16}$$

16.67MHz

$$t_{Srt-Rd} \geq 3 \tag{15}$$

$$t_{Srt} \geq 16 \tag{16}$$

863 FH PG 1028

15.1.4.3. Delay Settings

For 12.5 MHz operation using a delay line with taps every 2.5 nsec, .75 nsec tap to tap variation, and ± 1.5 nsec absolute variation requires placing Phi at 0, Rd at 12.5, Smp at 12.5, and Sys at 22.5.

For 16.67 MHz operation using a delay line with taps every 2 nsec, .5 nsec tap to tap variation, and ± 1 nsec absolute variation requires placing Phi at 0, Rd at 10, Smp at 10, and Sys at 18.

Cache Design

15.1.4.4. System Constraints

Using the delay settings as specified results in the following system constraints.

$$t_{DBus_{Hold_{Load}}} \geq t_{(Sys-Smp)_{Max}} - 7 \tag{12}$$

12.5 MHz
$$\geq 13 - 7$$

$$\geq 6$$

16.67 MHz
$$\geq 10 - 7$$

$$\geq 3$$

$$t_{DBus_{Hold_{Burst}}} \geq 5 - t_{(Smp-Rd)_{Min}} \tag{13}$$

12.5 MHz
$$\geq 5 - 0$$

$$\geq 5$$

16.67 MHz
$$\geq 5 - 0$$

$$\geq 5$$

$$t_{DBus_{Hold_{Xfer}}} \geq t_{(Sys-Smp)_{Max}} - 7 \tag{14}$$

12.5 MHz.
$$\geq 13 - 7$$

$$\geq 6$$

16.67 MHz
$$\geq 10 - 7$$

$$\geq 3$$

$$t_{SysClk_{Delay}} \geq t_{(Sys-Smp)_{Max}} - 5 \tag{18}$$

12.5 MHz
$$\geq 13 - 5$$
$$\geq 8$$

16.67 MHz
$$\geq 10 - 5$$
$$\geq 5$$

$$t_{Setup_{SysClk}} = 32.5 - t_{Sys_{Max}} \tag{19}$$

12.5 MHz
$$= 32.5 - 24$$
$$= 8.5$$

16.67 MHz
$$= 24 - 19$$
$$= 5$$

and.

$$t_{Hold_{SysClk}} = t_{(Sys-Rd)_{Min}} - 1 \tag{20}$$

12.5 MHz
$$= 7 - 1$$
$$= 6$$

16.67 MHz
$$= 6 - 1$$
$$= 5$$

The delay line settings, required data bus hold, and SysClk delay along with the resulting data setup and hold are summarized in the table below.

Parameter	12.5 MHz	16.67 MHz
Clk2xPhi	0	0
Clk2xRd	12.5	10
Clk2xSmp	12.5	10
Clk2xSys	22.5	18
$t_{DBus_{Hold}}$	6	5
$t_{SysClk_{Delay}}$	8	5
$t_{Setup_{SysClk}}$	8.5	5
$t_{Hold_{SysClk}}$	6	5

863 FH PG 1032

THE ADVANCED SYSTEMS OUTLOOK

Life Beyond RISC:
The next 30 years in high-performance computing

John P. Moussouris
Chairman & CEO • MicroUnity Systems

A HISTORY OF ERRORS — Predictions about high-performance computing have historically been subject to colossal blunders. Thirty or forty years ago, the worldwide market for electronic computer systems was estimated at six machines — a mistake comparable to estimating that the size the Great Wall of China would reach during its construction would be about six inches. It was a mistake that's observable at astronomical distances. So why did I commit to give a talk about a topic so insanely likely to give wrong predictions?

Basically, I saw this as an opportunity to make a very controversial contrarian prediction inspired by the malaise that I've seen in the electronics industry in recent years. It's being said that the electronics industry is now mature, that it's been a great ride for the past 30 years. We've seen one million times improvements in price performance, a kind of increase in price performance that has never been seen in any other industry in the recorded history of mankind, but that ride is now over. The technology is mature and we'll see much more gradual improvements in price performance over the next 30 years.

I predict that that's totally wrong, and that we will see an increase of a factor of a million in price performance in much less than 30 years. For those few of you who are skeptical, hear me now and believe me later.

THE GOOD OLD DAYS — Let's start by looking back at the last 30 years. Basically the history has been remarkably predictable and steady over the last 30 years. Every time semiconductor technology has made it possible to fit a general purpose computer into a more cost-effective package, a new family of computer systems has been born.

Back in the 1960s, the commercial mainframes were born fundamentally out of the earlier scientific machines — really the earliest machines were supercomputers. They were born with the arrival of transistor computing that made it possible to put an entire CPU in a single box that a commercial corporation could afford to buy and do useful work with. IBM really launched the heyday of the mainframe market with the 370 line (360 originally) that has continued for 25 years with a very steady exponential rate of growth.

Then in the 1970s integrated circuits arrived, and Digital Equipment Corporation and others started producing CPUs

This is an edited excerpt from an address at our fourth annual conference on the advanced systems outlook. The address was presented in San Francisco on June 5.

on a board. They came into their heyday with the DEC VAX, which has also had a very steady performance growth. The minicomputers were a lot cheaper than the mainframes, although with lower performance, and they co-existed with mainframes.

In the late 1970s the microprocessor became feasible with large scale integration. You had the entire CPU in a single chip at a much much lower cost but also lower performance, and that has continued forward with mainly compatible lines at a fairly predictable growth rate.

IT AIN'T NECESSARILY SO — The architectures that were designed to fit in the smaller package boundaries have suffered less from signaling delays and have been able to take advantage of scaling of improvements in the transistor technology better than the big machines, so they are growing at faster exponential rates. If you project this out to the early 1990s, there's a crossing, a kind of microunification that will take place where the micros outstrip minis and even mainframes.

And, in fact, an earlier version of this is occurring as compatibility with the old 1970s-era microprocessor architectures is abandoned in favor of the new reduced instruction set architectures, and there will be crossings of the mainframe lines very early in the 1990s if you look at the RISC growth curve (RISC being the machines that are designed to have the smallest CPU of all that can make the most efficient use of advanced transistor technology).

The significance of this crossing is that the last 30 years are of almost no use whatever in predicting the next 30 years in the computer industry. Clearly we will not be able to extrapolate these families which in the past have coexisted so peacefully another 30 years and have mainframes that are built at higher costs and which are slower and slower than low-cost microprocessors. Clearly that's not what's going to continue. On the other hand maybe we can get some insight by looking at the paradox of this singularity.

BELIEVE IT OR NOT — In the early days at MIPS (I was one of the co-founders at MIPS), I spent a lot of time explaining to people why RISC machines were fast. It was always difficult; people always were resistant to believe it. I remember someone telling me once that it was like a mechanic coming up to you opening up the hood of your car and starting to pull parts out and throw them away, doing that for a while and then closing the hood and telling you to get in and start the car up, all the while claiming that the car is going to run faster, consume less gasoline and be more reliable — that's just going to be really hard to believe. But the truth is that now it's accepted by even Motorola and Intel that RISC works — they take full page ads in *The Wall Street Journal* to advertise that fact.

And, there's a way to look at it that's so so paradoxical. In RISC there is a certain size of CPU which is the optimum size, given a particular semiconductor technology, for building the fastest possible general purpose processor. If you try to build a processor that has more hardware than that optimum amount, not only will it cost more, but it will actually be slower.

Around 1970, the fastest CPU you could build (which was a Cray, basically) required over 100,000 logic chips just for the logic part of the CPU. I predict that in the early 1990s the fastest general purpose CPU will be a single chip RISC microprocessor. And if you project this trend forward and assume even fairly conservative improvements in semiconductor density, 20 years later in the year 2010 it will not be possible to make a general purpose CPU any faster by covering more than 1/100th the area of an economically manufacturable single chip.

THANKS FOR THE MEMORY — If that's the case, what will we do with the other 90% of the chip? Well it turns out we'll do something quite similar to what we've done in the past; we will basically use it for memory.

In these successive generations of machines, the key factor is that an execution unit, to get a lot of work done, needs to access lots of information from memory. The rate at which it can do work is gated by the rate it can access information from memory — the bandwidth of access to memory. You exploit the fact that small memories are fast to organize storage in a hierarchy with very small fast memory close to the execution unit and then successively slower bigger memories as you go farther away. You're using the way programs handle memory to take advantage of the fact that most of the time you can satisfy the needs for information from the smaller memories, and only seldom do you have to go out to the bigger memories. That's the key trick that all machines use to overcome the bottleneck between the execution unit and memory.

A single chip in a mainframe is just a small part of the execution unit. In a minicomputer, it is a bigger piece of the execution unit. Micros, however, made a tremendous leap forward by including the entire central processor unit, including the registers, on the single chip. What that means is that if you have a given amount of data transfer capability at the chip boundary, you can do a lot more work in the execution units because most of the needs for information are satisfied in the registers.

CACHE ME, IF YOU CAN — Now RISC machines went one better than traditional micros by having a lot more registers, so they work a lot more efficiently (especially if you have the right software) and also by including more and more of the cache, the next level of the memory hierarchy. I predict that the event that will cause RISC machines to become the fastest general purpose CPUs on the planet will be the inclusion of essentially all the cache on the chip. And you'll see that in the early 1990s from almost all the RISC vendors; the technology is there.

The progression in the future will be to continue this process to include a very large amount of main memory on the chip along with all of the rest of the central processing unit structure. When you're doing is alleviating bandwidth bottlenecks; you're raising the ability of the processor to get its hands on the information it needs to do useful work. That's where the performance is coming from and why these small machines that can include more of the hierarchy on the chip do better and grow faster in speed.

STRIKE UP THE BANDWIDTH — So the real secret behind RISC is bandwidth. If a master of RISC design were on his deathbed and his prodigal son came to him (finally realizing that he'd better get the secret before Pop dies so that he can go into the business himself), and the old man

and only one word left to impart all of his wisdom into his prodigal son, that one word would be "Bandwidth."

A constant of nature in computing is that it takes about 10 bytes of information to do a typical operation, so to get a million instructions per second you have to have a bandwidth of about 10 megabytes per second between the execution units and the memory hierarchy. If you look at current RISC processors, their performance is very accurately predictable by the bandwidth of the chip boundaries. For example, presently the highest bandwidth is the MIPS R3000 which fetches 64 bits of information across its boundary in a single cycle. If you looked at the Sun SPARC implementation or the Intel i860, it takes two cycles to get 64 bits. And for real programs that don't fit into small caches on the chip or that don't just work out of the register file, the performance is pretty much proportional to those cycle counts.

GAPOSIS — Now the thing that's holding the industry back is that the standard interfaces being provided by chip vendors have improved in bandwidth incredibly slowly relative to the underlying transistor improvements.

Every three years for the last twenty years, DRAM density has improved by a factor of four — we've gone from one kilobit DRAM in 1971 to four megabit DRAM in 1990. The bandwidth of a four megabit DRAM is only about five times as great as the old 20 year old part, and a gap of a factor of 250 has opened up. There's no technical reason for this; the reason is the marketing pressure for compatibility between successive generations of chips.

However, that gap has gotten so gigantic, and there are such gigantic commercial pressures to close it, that finally new standards are beginning to emerge that will close this gap and ease the pain of moving to non-compatible hardware. For example there is an IEEE working group called P1596 that has been participated in for over a year now by a number of companies — HP, Motorola, MIPS, Sequent, Apple — to produce a new standard which is based on extremely fast signalling technology — five hundred million transfers per second, two bytes per transfer across an in link and an out link. So two bytes times a half a billion transfers per second is a gigabyte per second of bandwidth in each of two directions across a fairly small number of wires.

In fact if you took this kind of signalling technology and applied it to a 40-pin surface-mount RAM package, you would exactly close the gap. This is specifically geared for scalable multiprocessors, scaling up to large processor count with a coherent memory model, so it's called the Scalable Coherent Interface. Because bandwidth is especially important for microprocessors, high bandwidth standards are especially important for multiprocessors. This is a response to that need.

POWER SURGE — Now, based on this, I want to make a bunch of hardware predictions. First of all, I believe that over the next decades, multiprocessors will dominate. We'll see product lines all based on microprocessors, but small machines will have one or two microprocessors in them, medium-sized machines will have 10 microprocessors in them, and large machines will have hundreds or even thousands of microprocessors in them.

Also, because of the very high bandwidth required for these

multiprocessors, we will see terabits per second (trillions of bits per second) of bandwidth cost-reduced to the point where they appear on desktops. Present desktop machines get less than a gigabit per second of bandwidth. We're talking about an improvement of more than 1,000 times the bandwidth available in a desktop at constant cost.

Another phenomenon you'll see, as even the very fastest processor can't take up more than 10% of a chip, is that processors will start appearing on almost every kind of chip. Even fast RAMS will have processors included on them. They will have programmable intelligence included on them. Initially for traffic cop and diagnostic purposes, but eventually for doing real compute work as well.

IT'S ALL IN THE SOFTWARE — Now the gating factor in all of this is going to be the software, of course. We've already seen with RISC that, as very powerful versatile processors come into existence, software ends up replacing functions that have historically been done with hardware. In RISC machines you take out the microcode and put optimizing compilers in instead. You take out the hardwired memory management unit and put in a programmable system co-processes. In the future, you'll take out the protocol hardware and put in programmable interface processors in multiprocessor configurations.

The vast majority of the software will be written in high-level languages. Open systems will dominate. You will not see new proprietary systems; large amounts of code written in assembly will not exist in the future. The old proprietary architectures will dwindle over the coming years. And the prime virtue for new software is going to be multiprocessor transparency — the ability to run not just on different processors but different numbers of processors without intervention.

TRILLIONS AND TRILLIONS — I believe that in only 15 years we will see a million times improvement in price performance. A $10,000 workstation in the year 2005 will have a few hundred chips in it just like it has today. Those chips will cost about the same to manufacture as they do today except that most of those chips will have not only a substantial amount of memory on them but also a processor capable of a few billion ops per second. And a few hundred chips times a giga ops per chip is a trillion ops — a tera op on a desktop for about $10,000, as compared to today's giga op in a supercomputer for $10 million. That's a million times improvement in price performance.

And this will not require any fundamental advances - it won't require the memory technology to continue to accelerate even as fast as it has historically during the last 20 years. For example, this would be feasible with 32 megabytes of memory per chip. That only requires three step-ups, three quadruplings which would happen in nine years at the present rate. And I'm assuming it will happen in 15 years: that's quite conservative.

Also, the chip interfaces could just be the P1596 style interfaces, and with 32 megabytes of memory on each chip, there would be adequate bandwidth to connect together a few hundred like this. So nothing very exotic is required for this to happen.

EXOTICA — On the other hand there's lots of headroom in the technology for exotic things to happen. For example,

optical interconnect is a factor of a million away from its physical limits. It's incredibly primitive today. Ordinary visible light and fiber has frequencies of about 1000 terahertz, or 1000 terabits per second of potential bandwidth, and we presently use about a gigabit per second.

There's as big a jump between what we do today and what could be done as there is between fire and fusion. There's all this excitement about fusion but much less exotic technologies would make this jump accessible to the computer industry.

ON WITH THE SHOW — Now, who cares about all this improvement? What's the market going to do with it? How can the market respond to it if there's no appetite for it? I think there's an insatiable appetite for improvements in this technology. Bandwidth is the important item, not megabytes or chips.

Today, digital audio is about the limit of what we can do programmably in a general purpose microprocessor, and it requires a couple megabits per second of bandwidth. We're beginning now to talk about digital HDTV, and that requires about a gigabit per second. Digital cinema requires about 10 times as much resolution as HDTV in each direction to be as good in imaging as digital audio is on the audio side. That's about 100 gigabits per second.

If you try to have 1000 people in a large room wearing stereo vision and sharing a 3-dimensional high resolution audio visual environment that responds to changes in where they look and changes in an underlying model of the reality — it could be in a movie or it could be in a war games simulation — looking at just synthetic reality, you need another factor of 1000, or 100 terabits per second of bandwidth.

I predict that as the technology becomes available for the tera op workstations, this kind of bandwidth is going to become extremely cost-effective and will become everyday reality. Theaters will have digital cinema and then synthetic reality. War games will be done this way — not by destroying real equipment and hazarding lives.

ALL YOU CAN EAT — So there's lots and lots of need for this bandwidth. In the end, bandwidth is going to be the valuable resource. Computers are already outliving their initial role as calculating engines and their subsequent role as thinking machines. And I think in the end their economically most significant and socially most significant role will be as intelligent media. Bandwidth is the technological measure of intensity of experience — how long it takes you to get information. That's the ultimate gating item of human experience and of productivity in the workplace. So I believe that the appetite for this bandwidth is essentially insatiable.

In conclusion, RISC is here to stay; there will be lots more of it. The software that will be most significant over the next few decades will be open systems with multiprocessor transparency. I believe that in 15 years we will see a million times improvement in price performance — much less than the 30 years it took us to get where we are now — that bandwidth will set the pace for this advance, and that the technology has lots of headroom. So, there's plenty of life beyond RISC, and the next 30 years in high-performance computing will be even more exciting than the first 30. ▪

Life Beyond RISC:
The Next Thirty Years in
High Performance Computing

John P. Moussouris

MicroUnity Systems

The Last Thirty Years
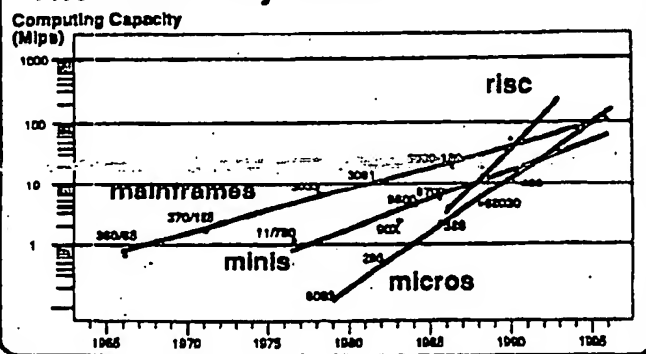
MicroUnity Systems

The RISC Paradox

Just enough CPU beats too much

How much is enough?

    1970: >100,000 chip CRAY
    1990: 1 chip RISC micro
    2010: <0.1 chip CPU node

MicroUnity Systems

INTEGRATION
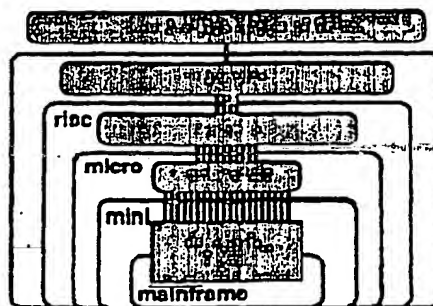
MicroUnity Systems

Bandwidth

Bandwidth sets the pace

10 bytes / op

i.e. 10 Mbytes/sec per mip

MicroUnity Systems

THE BANDWIDTH GAP
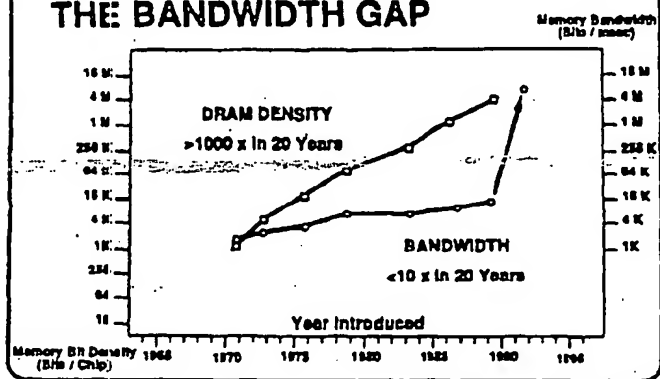
MicroUnity Systems

Scalable Coherent Interface (IEEE P1596)

```
        ┌─────────┐
        │   SCI   │
        │  NODE   │
        └─────────┘
    17x2 ↑      ↓ 17x2
        IN     OUT
```
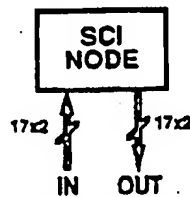
Differential ECL at 500 MHz

Hardware Predictions

Multiprocessors will dominate

Terabits/sec on the desktop

**A processor in every chip
- even RAM!**

Software Predictions

Software replaces hardware

Open systems dominate

MP Transparency

MicroUnity Systems

WorkStation ca. 2005

A few hundred chips

X a few GigaOps per chip

= 1 TeraOp for <$10,000

MicroUnity Systems

Headroom

Information Capacity of a single
Optical Fiber = 1000 Terabits/sec

MicroUnity Systems

Appetite for Bandwidth

Digital Audio: 1.6 Mbits/sec
Digital HDTV: 1 Gbit/sec
Digital Cinema: 100 Gbits/sec
Synthetic Reality: 100 Tbits/sec

MicroUnity Systems

Conclusions

RISC shall be fruitful and multiply

Open systems MP transparent

1,000,000 X in 15 years

Bandwidth sets the pace

Technology has lots of headroom

MicroUnity Systems

MicroUnity

ultra high bandwidth digital systems

MicroUnity Systems

(12) **United States Patent**
Farmwald et al.

(10) Patent No.: **US 6,452,863 B2**
(45) Date of Patent: ***Sep. 17, 2002**

(54) **METHOD OF OPERATING A MEMORY DEVICE HAVING A VARIABLE DATA INPUT LENGTH**

(75) Inventors: **Michael Farmwald**, Berkeley; **Mark Horowitz**, Palo Alto, both of CA (US)

(73) Assignee: **Rambus Inc.**, Los Altos, CA (US)

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 169 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **09/492,982**

(22) Filed: **Jan. 27, 2000**

Related U.S. Application Data

(63) Continuation of application No. 09/252,997, filed on Feb. 19, 1999, now Pat. No. 6,034,918, which is a continuation of application No. 09/196,199, filed on Nov. 20, 1998, now Pat. No. 6,038,195, which is a continuation of application No. 08/798,520, filed on Feb. 10, 1997, now Pat. No. 5,841,580, which is a division of application No. 08/448,657, filed on May 24, 1995, now Pat. No. 5,638,334, which is a division of application No. 08/222,646, filed on Mar. 31, 1994, now Pat. No. 5,513,327, which is a continuation of application No. 07/954,945, filed on Sep. 30, 1992, now Pat. No. 5,319,755, which is a continuation of application No. 07/510,898, filed on Apr. 18, 1990, now abandoned.

(51) Int. Cl.[7] G11C 8/00

(52) U.S. Cl. 365/233; 365/189.01

(58) Field of Search 365/233, 235, 365/238.5; 39/189.01

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,691,534 A	9/1972	Veradi et al.	365/78
3,771,145 A	11/1973	Wiener	365/240
3,882,470 A	5/1975	Hunter	365/200

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

EP	0 246 767	4/1987
EP	0 276 871	1/1988
EP	0282735	9/1988

(List continued on next page.)

OTHER PUBLICATIONS

M. Bazes et. al., "A Programmable NMOS DRAM Controller for Microcomputer Systems with Dual–Port Memory and Error Checking and Correction", IEEE Journal of Solid State Circuits, vol. 18 No. 2, pp. 164–172 (Apr. 1983).
A. Agarwal, "An Evaluation of Directory Schemes for Cache Coherence", IEEE document pp. 280–289 (1988).
D. Kawley, "Superfast Bus Supports Superfast Transactions", High Performance Systems, pp. 90–94 (Sep. 89).

(List continued on next page.)

Primary Examiner—Tan T. Nguyen
(74) *Attorney, Agent, or Firm*—Neil A. Steinberg

(57) **ABSTRACT**

A method of controlling a memory device, wherein the memory device includes a plurality of memory cells. The method includes providing first block size information to the memory device, wherein the first block size information defines a first amount of data to be input by the memory device in response to a write request. The method further includes issuing a write request to the memory device, wherein in response to the write request the memory device inputs the first amount of data corresponding to the first block size information.

35 Claims, 14 Drawing Sheets

REGULAR ACCESS
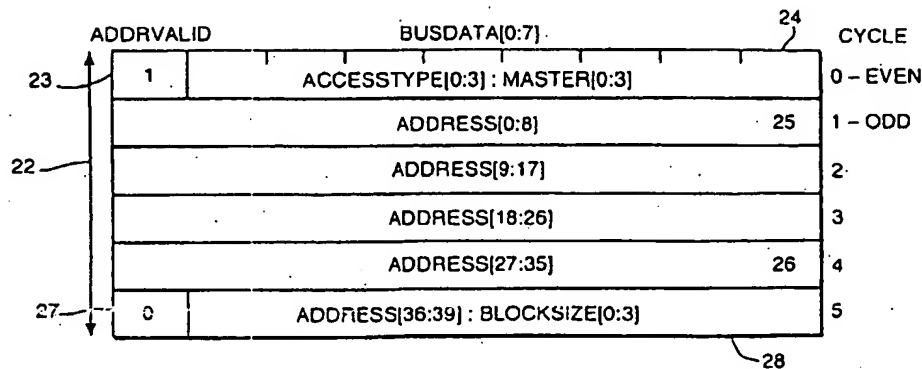
U.S. PATENT DOCUMENTS

4,047,246 A	9/1977	Kerllenevich et al.	710/61
4,048,673 A	9/1977	Hendrie et al.	710/129
4,092,665 A	5/1978	Saran	341/63
4,099,231 A	7/1978	Kotok et al.	711/168
4,183,095 A	1/1980	Ward	365/189.02
4,205,373 A	5/1980	Shah et al.	710/128
4,231,104 A	10/1980	St. Clair	713/500
4,330,852 A	5/1982	Redwine et al.	365/221
4,337,523 A	6/1982	Hotta et al.	365/194
4,394,753 A	7/1983	Penzel	365/236
4,435,762 A	3/1984	Milligan et al.	710/6
4,445,204 A	4/1984	Nishiguchi	365/194
4,466,127 A	8/1984	Ohgishi et al.	455/182.01
4,482,999 A	11/1984	Janson et al.	370/452
4,509,142 A	4/1985	Childers	364/900
4,513,370 A	4/1985	Ziv et al.	709/253
4,536,795 A	8/1985	Hirota et al.	348/714
4,566,099 A	1/1986	Magerl	370/509
4,586,167 A	4/1986	Fujishima et al.	365/189.05
4,589,108 A	5/1986	Billy	370/503
4,616,268 A	10/1986	Shida et al.	358/451
4,625,307 A	11/1986	Tulpule et al.	370/402
4,629,909 A	12/1986	Cameron	327/211
4,631,659 A	12/1986	Hayne et al.	711/167
4,648,102 A	3/1987	Riso et al.	375/356
4,663,735 A	5/1987	Novak et al.	345/515
4,672,470 A	6/1987	Morimoto et al.	386/16
4,675,850 A	6/1987	Kumanoya et al.	365/189.08
4,680,738 A	7/1987	Tam	365/239
4,685,088 A	8/1987	Ianucci	365/194
4,703,418 A	10/1987	James	364/200
4,719,505 A	1/1988	Katznelson	348/502
4,719,602 A	1/1988	Hag et al.	365/189.02
4,726,021 A	2/1988	Horiguchi et al.	371/38
4,734,880 A	3/1988	Collins	711/105
4,748,617 A	5/1988	Drewlo	359/121
4,750,839 A	6/1988	Wang et al.	365/233
4,755,937 A	7/1988	Glier	
4,763,249 A	8/1988	Bomba et al.	364/200
4,785,394 A	11/1988	Fischer	710/114
4,785,428 A	11/1988	Bajwa	365/233
4,788,667 A	11/1988	Nakano et al.	365/193
4,792,926 A	12/1988	Roberts	365/189.02
4,799,199 A	1/1989	Scales, III et al.	365/230.08
4,803,621 A	2/1989	Kelly	711/5
4,807,189 A	2/1989	Pinkham et al.	365/189.05
4,821,226 A	4/1989	Christopher et al.	365/230.03
4,825,287 A	4/1989	Baji et al.	348/720
4,825,416 A	4/1989	Tam et al.	365/194
4,835,674 A	5/1989	Collins et al.	709/214
4,839,801 A	6/1989	Nicely et al.	710/35
4,845,664 A	7/1989	Aichelmann, Jr. et al.	364/900
4,845,670 A	7/1989	Nishimoto et al.	365/78
4,845,677 A	7/1989	Chappell et al.	365/189.02
4,849,965 A	7/1989	Chomel et al.	370/438
4,851,990 A	7/1989	Johnson et al.	710/100
4,870,562 A	9/1989	Kimoto et al.	364/200
4,870,622 A	9/1989	Aria et al.	365/230.02
4,873,671 A	10/1989	Kowshik et al.	365/189.12
4,875,192 A	10/1989	Matsumoto	
4,876,670 A	10/1989	Nakabayashi et al.	365/189.12
4,878,166 A	10/1989	Johnson et al.	710/127
4,882,712 A	11/1989	Ohno et al.	365/206
4,891,791 A	1/1990	Iijima	365/189.01
4,901,036 A	2/1990	Herold et al.	331/25
4,920,483 A	4/1990	Pogue et al.	364/200
4,926,385 A	5/1990	Fujishima et al.	365/230.03
4,928,265 A	5/1990	Beighe et al.	365/189.01
4,937,734 A	6/1990	Bechtolsheim	711/202
4,945,516 A	7/1990	Kushiyama	365/189.05

4,949,301 A	8/1990	Joshi et al.	711/100
4,951,251 A	8/1990	Yamaguchi et al.	365/189.02
4,953,128 A	8/1990	Kawai et al.	365/194
4,954,987 A	9/1990	Auvinen et al.	365/189.02
4,975,872 A	12/1990	Zaiki	365/49
4,979,145 A	12/1990	Remington et al.	711/106
5,009,481 A	4/1991	Kinoshita et al.	385/33
5,016,226 A	5/1991	Hiwada et al.	365/233
5,018,111 A	5/1991	Madland	365/233
5,029,124 A	7/1991	Leahy et al.	710/105
5,034,964 A	7/1991	Khan et al.	375/242
5,036,495 A	7/1991	Busch et al.	365/233
5,083,260 A	1/1992	Tsuchiya	710/113
5,083,296 A	1/1992	Hara et al.	365/230.02
5,093,807 A	3/1992	Hashimoto et al.	365/230.09
5,107,465 A	4/1992	Fung et al.	365/230.08
5,109,498 A	4/1992	Kamiya et al.	395/425
5,111,486 A	5/1992	Oliboni et al.	375/120
5,123,100 A	6/1992	Hisada et al.	713/401
5,134,699 A	7/1992	Aria et al.	710/35
5,140,688 A	8/1992	White et al.	345/550
5,142,378 A	8/1992	Ogura	386/29
5,142,637 A	8/1992	Harlin et al.	345/425
5,148,523 A	9/1992	Harlin et al.	345/519
5,175,835 A	12/1992	Beighe et al.	711/212
5,179,667 A	1/1993	Iyer	711/167
5,193,193 A	3/1993	Iyer	710/117
5,206,833 A	4/1993	Lee	365/233
5,276,846 A	1/1994	Aichelmann, Jr. et al.	711/165
5,301,278 A	4/1994	Bowater et al.	711/5
5,361,277 A	11/1994	Grover	375/36
5,684,753 A	11/1997	Hashimoto et al.	365/233
6,034,918 A *	3/2000	Farmwald et al.	365/233

FOREIGN PATENT DOCUMENTS

EP	0 334 552	3/1989
EP	0218523	5/1989
EP	0449052	3/1990
EP	0424774	5/1991
JP	S56-82961	7/1981
JP	S57-14922	1/1982
JP	Sho 60-80193	5/1983
JP	SHO 58-192154	11/1983
JP	Sho 60-55459	3/1985
JP	S61-72350	4/1986
JP	SHO 61-72350	5/1986
JP	SHO 61-160556	10/1986
JP	SHO 62-16289	1/1987
JP	62-51509	3/1987
JP	SHO 63-34795	2/1988
JP	SHO 63-91766	4/1988
JP	S63-142445	6/1988
JP	B63-46864	9/1988
JP	S64-29951	1/1989
WO	WO 89/12936	12/1989

OTHER PUBLICATIONS

H. L. Kalter et al., "A 50-ns 16Mb DRAM with a 10-ns Data Rate and On-Chip ECC", IEEE Journal of Solid State Circuits, vol. 25 No. 5, pp. 1118–1128 (Oct. 1990).

S. Watanabe et. al., "AN Experimental 16-Mbit CMOS DRAM Chip with a 100-MHz serial READ/WRITE Mode", IEEE Journal of Solid State Circuits, vol. 24 No. 3, pp. 763–770 (Jun. 1982).

T.L. Jeremiah et. al., "Synchronous Packet Switching Memory and I/O Channel," IBM Tech. Disc. Bul., vol. 24, No. 10, pp. 4986–4987 (Mar. 1982).

L. R. Metzeger, "A 16K CMOS PROM with Polysilicon Fusible Links", IEEE Journal of Solid State Circuits, vol. 18 No. 5, pp. 562–567 (Oct. 1983).

A. Yuen et. al., "A 32K ASIC Synchronous RAM Using a Two-Transistor Basic Cell", IEEE Journal of Solid State Circuits, vol.24 No. 1, pp. 57–61 (Feb. 1989).

D.T. Wong et. al., "An 11-ns 8K×18 CMOS Static RAM with 0.5µm Devices", IEEE Journal of Solid State Circuits, vol. 23 No. 5, pp. 1095–1103 (Oct. 1988).

T. Williams et. al., "An Experimental 1-Mbit CMOS SRAM with Configurable Organization and Operation", IEEE Journal of Solid State Circuits, vol. 23 No. 5, pp. 1085–1094 (Oct. 1988).

D. Jones, "Synchronous static ram", Electronics and Wireless World, vol. 93, No. 1622, pp. 1234–1244 (Dec. 87).

F. Miller et. al., "High Frequency System Operation Using Synchronous SRAMS", Midcon/87 Conference Record, pp. 430–432 Chicago, IL, USA; Sep. 15–17, 1987.

K. Ohta, "A 1-Mbit DRAM with 33-MHz Serial I/O Ports", IEEE Journal of Solid State Circuits, vol. 21 No. 5, pp. 649–654 (Oct. 1986).

K. Nogami et. al., "A 9-ns HIT-Delay 32-kbyte Cache Macro for High-Speed RISC", IEEE Journal of Solid State Circuits, vol. 25 No. 1, pp. 100–108 (Feb. 1990).

F. Towler et. al., "A 128k 6.5ns Access/5ns Cycle CMOS ECL Static RAM", 1989 IEEE international Solid State Circuits Conference, (Feb. 1989).

M. Kimoto, "A 1.4ns/64kb RAM with 85ps/3680 Logic Gate Array", 1989 IEEE Custom Integrated Circuits Conference.

D. Wendell et. al. "A 3.5ns, 2K×9 Self Timed SRAM", 1990 IEEE Symposium on VLSI Circuits (Feb. 1990).

R. Schmidt, "A memory Control Chip fo Formatting Data into Blocks Suitable for Video Applications", IEEE Transactions on Circuits and Systems, vol. 36, No. 10 (Oct. 1989).

D. K. Morgan "The CVAX CMCTL—A CMOS Memory Controller Chip", Digital Technical Journal, No. 7 (Aug. 1988).

T.C. Poon et. al., "A CMOS DRAM-Controller Chip Implementation", IEEE Journal of Solid State Circuits, vol. 22 No. 3, pp. 491–494 (Jun. 1987).

K. Numata et. al. "New Nibbled-Page Architecture for High Density DRAM's", IEEE Journal of Solid State Circuits, vol. 24 No. 4, pp. 900–904 (Aug. 1989).

E.H. Frank "The SBUS: Sun's High Performance System Bus for RISC Workstations" Sun Microsystems Inc. 1990.

Watanabe, T.; "Session XIX: High Density SRAMS"; IEEE International Solid State Circuits Conference pp. 266–267 (1987).

Ohno, C.; "Self-Timed RAM: STRAM"; Fujitsu Sci. TechJ., 24, 4, pp. 293–300 (Dec. 1988).

"Fast Packet Bus for Microprocessor Systems with Caches", IBM Technical Disclosure Bulletin, pp. 279–282 (Jan. 1989).

Gustavson, D. "Scalable Coherent Interface"; Invited Paper, COMPCON Spring '89, San Francisco, CA; IEEE, pp. 536–538 (Feb. 27–Mar. 3, 1989).

James, D.; "Scalable I/O Architecture for Busses"; IEEE, pp. 539–544 (Apr. 1989).

European Search Report for EPO Patent Application No. 00 101 1832.

European Search Report for EPO Patent Application No. 89 30 2613.

Z. Amitai, "New System Architectures for DRAM Control and Error Correction", Monolithic Memories Inc., Electro/87 and Mini/Mico Northeast: Focusing on the OEM Conference Record, pp. 1132, 4/31–3, (Apr. 1987).

N. Siddique, "100-MHz DRAM Controller Sparks Multiprocessor Designs", Electronic Design, pp. 138–141, (Sep. 1986).

H. Kuriyama et al., "A 4-Mbit CMOS SRAM with 8-NS Serial Access Time", IEEE Symposium On VLSI Circuits Digest Of Technical Papers, pp. 51–52, (Jun. 1990).

J. Chun et al., "A 1.2ns GaAs 4K Read Only Memory", IEEE Gallium Arsenide Integrated Circuit Symposium Technical Digest, pp. 83–86, (Nov. 1988).

A. Fielder et al., "A 3 NS 1K X 4 Static Self-Timed GaAs RAM", IEEE Gallium Arsenide Integrated Circuit Symposium Technical Digest, pp. 67–70, (Nov. 1988).

JEDEC Standard No. 21C.

Takasugi, A. et al., "A Data-Transfer Architecture for Fast Multi-Bit Serial Acess Mode DRAM," 11th European Solid State Circuits Conference, Toulouse, France pp. 161–165 (Sep. 1985).

Amitai, Z., "Burst Mode Memories Improve Cache Design," WESCON/90 Conference Record, pp. 279–282 (Nov. 1990).

Ikeda, Hiroaki et al., "100 MHz Serial Acess Architecture for 4Md Field Memory," Symposium of VLSI Circuits, Digest of Technical Papers, pp. 11–12 (Jun. 1990).

Schmitt-Landsiedel, Doris, "Pipeline Architecture for Fast CMOS Buffer RAMs," IEEE Journal of Solid-State Circuits, vol. 25, No. 3, pp. 741–747 (Jun. 1990).

Horowitz et al., "MIPS-X: A 20-MIPS Peak 32-Bit Microprocessor with ON-Chip Cache", IEEE J. Solid State Circuits, vol. SC-22, No. 5, pp. 790–798 (Oct. 1987).

Robert J. Lodi et al., "Chip and System Characteristics of a 2048-Bit MNOS-BORAM LSI Circuit," 1976 IEEE International Solid-State Circuits Conference (Feb. 18, 1976).

Whiteside, Frank, "A Dual-Port 65ns 64K×4 DRAM with a 50MHz Serial Output,"IEEE International Solid-State Circuits Conference Digest (Feb. 1986).

Pinkham, Raymond, "A High Speed Dual Port Memory with Simultaneous Serial and Random Mode Access for Video Applications," IEEE Journal of Solid-State Circuits, vol. SC-19, No. 6, pp. 999–1007 (Dec. 1984).

Ishimoto, S. et al., "a 256K Dual Port Memory," ISSCC Digest of Technical Papers, pp. 38–39 (Feb. 1985).

Iqbal, Mohammad Shakaib, "Internally Timed RAMs Build Fast Writable Control Stores," Electronic Design, pp. 93–96 (Aug. 25, 1988).

Schnaitter, William M. et al., "A 0.5-GHz CMOS Digital RF Memory Chip," IEEE Journal of Solid-State Circuits, vol. SC-21, No. 5, pp. 720–726 (Oct. 1986).

Bursky, Dave, "Advanced Self-Timed SRAM Pares Access Time to 5 ns," Electronic Design, pp. 145–147 (Feb. 22, 1990).

Tomoji Takada et al., "A Video Codec LSI for High-Definition TV Systems with One-Transistor DRAM Line Memories," IEEE Journal of Solid-State Circuits, vol. 24, No. 6, pp. 1656–1659 (Dec. 1989).

Robert J. Lodi et al., "MNOS-BORAM Memory Characteristics," IEEE Journal of Solid-State Circuits, vol. SC-11, No. 5, pp. 622–631 (Oct. 1976).

Gregory Uvieghara et al., "an On-Chip Smart Memory for a Data-Flow CPU," IEEE Journal of Solid-State Circuits, vol. 25, No. 1, pp. 84–89 (Feb. 1990).

Ray Pinkham et al., "A 128Kx8 70-MHz Multiport Video RAM with Auto Register Reload and 8x4 WRITE Feature," IEEE Journal of Solid State Circuits, vol. 23, No. 3, pp. 1133–1139 (Oct. 1988).

Hans–Jurgen Mattausch et al., "a Memory–Based High–Speed Digital Delay Line with a Large Adjustable Length," IEEE Journal of Solid–State Circuits, vol. 23, No. 1, pp. 105–110 (Feb. 1988).

Kanopoulos, Nick and Jill H. Hallenbeck, "A First–In, First–Out Memory for Signal Processing Applications," IEEE Transactions on Circuits and Systems, vol. CAS–33, No. 5, pp. 556–558 (May 1986).

Pelgrom et al., "A 32–kbit Variable–Length Shift Register for Digital Audio Application", IEEE Journal of Solid–State Circuits, vol. sc–22, No. 3, Jun. 1987, pp. 415–422.

Grover et al., "Precision Time–Transfer in Transport Networks Using Digital Crossconnect Systems", IEEE Paper 47.2 Globecom, 1988, pp 1544–1548.

Gustavson et al., "The Scalable Interface Project (Super-bus)" (DRAFT), SCI–22 Aug 88–doc1 pp. 1–16, Aug. 22, 1988.

Knut Alnes, "SCI: A Proposal for SCI Operation", SCI–10Nov88–doc23, Norsk Data, Oslo, Norway, pp. 1–12, Nov. 10, 1988.

Knut Alnes, "SCI: A Proposal for SCI Operation", SCI–6Jan89–doc31, Norsk Data, Oslo, Norway, pp. 1–24, Jan. 6, 1989.

Bakka et al., "SCI: Logical Level Proposals", SCI–6Jan89–doc32, Norsk Data, Oslo, Norway, pp. 1–20, Jan. 6, 1989.

Knut Alnes, "Scalable Coberent Interface", SCI–Feb89–doc52, (To appear in Eurobus Conference Proceedings May 1989) pp. 1–8.

Boysel et al., "Four–Phase LSI Logic Offers New Approach to Computer Designer", Four–Phase Systems Inc. Cupertino, CA, Computer Design, Apr. 1970, pp. 141–146.

Boysel et al., "Random Access MOS Memory Packs More Bits To The Chip", Electronics, Feb. 16, 1970, pp. 109–146.

Hansen et al., "A RISC Microprocessor with Integral MMU and Cache Interface", MIPS Computer Systems, Sunnyvale, CA, IEEE 1986 pp. 145–148.

Moussouris et al., "A CMOS Processor with Integrated Systems Functions", MIPS Computer Systems, Sunnyvale, CA, IEEE 1986 pp. 126–130.

"LR2000 High Performance RISC Microprocessor Preliminary" LSI Logic Corp. 1988, pp. 1–15.

"LR2010 Floating Point Accelerator Preliminary" LSI Logic Corp. 1988, pp. 1–20.

"High Speed CMOS Databook", Integrated Device Technology Inc. Santa Clara, CA, 1988 pp. 9–1 to 9–14.

Riordan T. "MIPS R2000 Processor Interface 78–00005(C)", MIPS Computer Systems, Sunnyvale, CA, Jun. 30, 1987, pp. 1–83.

Moussouris, J. "The Advanced Systems Outlook–Life Beyond RISC: The next 30 years in high–performance computing", Computer Letter, Jul. 31, 1989 (an edited excerpt from an address at the fourth annual conference on the Advanced Systems Outlook, in San Francisco, CA (Jun. 5)).

* cited by examiner

FIG_1

RAM ARRAY — 1

ROW DECODE

WORD LINES — 5

BITLINES

6

COLUMN AMPLIFIERS — 3

COLUMN SELECT

FIG_2

FIG_3

19

20

18

15

16

17

REGULAR ACCESS

FIG_4

ADDRVALID	BUSDATA[0:7]	CYCLE
1	ACCESSTYPE[0:3] : MASTER[0:3]	0 – EVEN
		1 – ODD
	ADDRESS[0:8]	2
	ADDRESS[9:17]	3
	ADDRESS[18:26]	
	ADDRESS[27:35]	4
0	ADDRESS[36:39] : BLOCKSIZE[0:3]	5

24

25

26

28

23

22

27
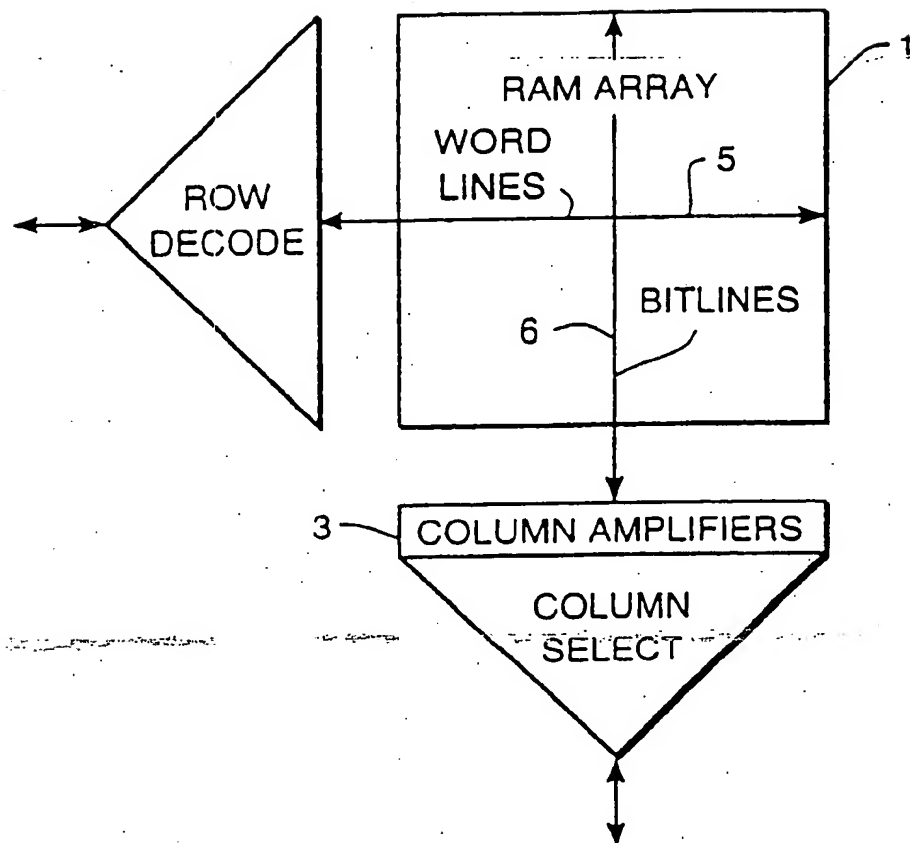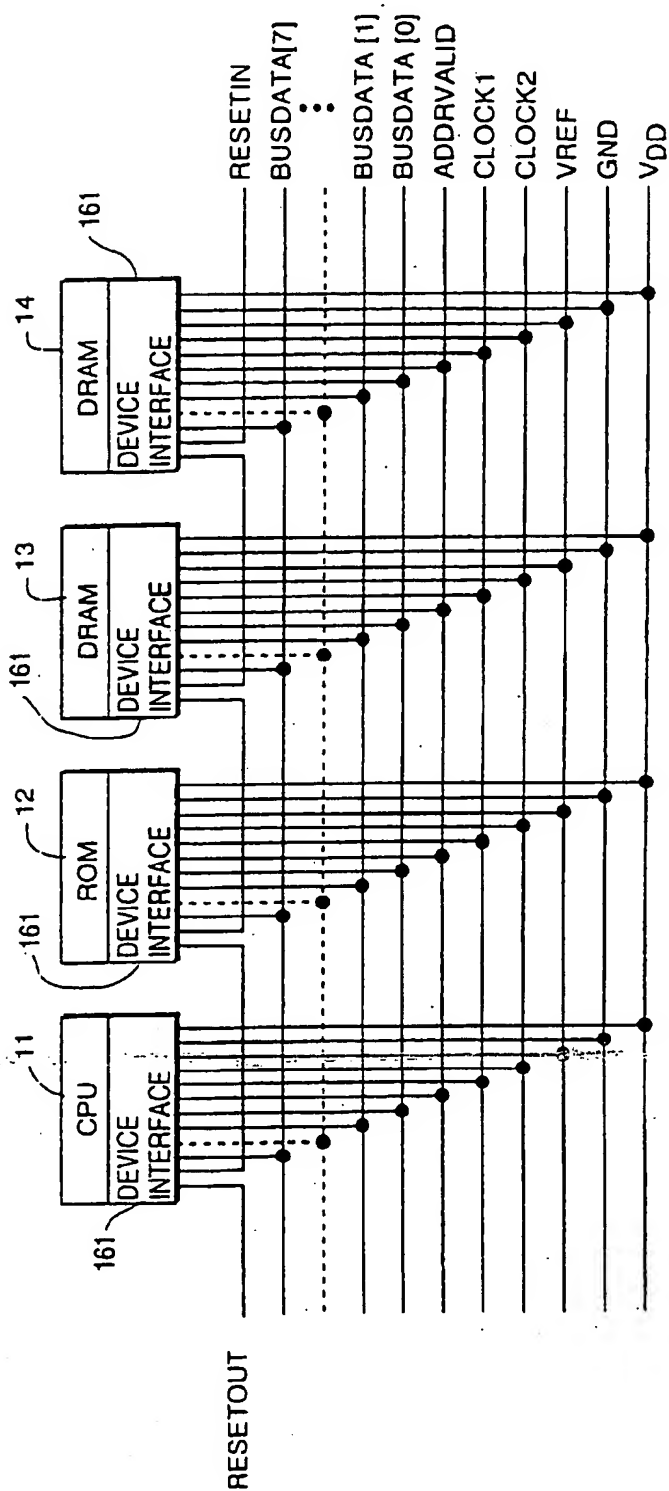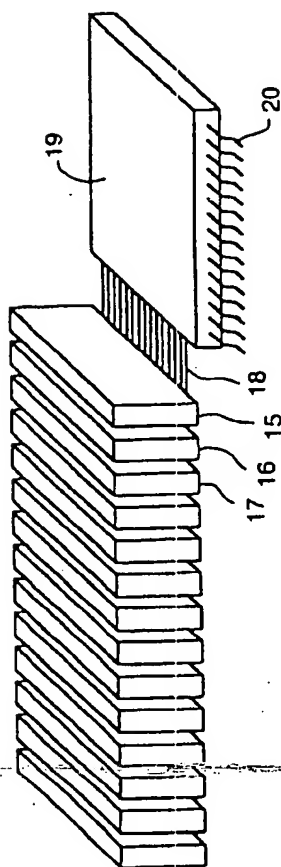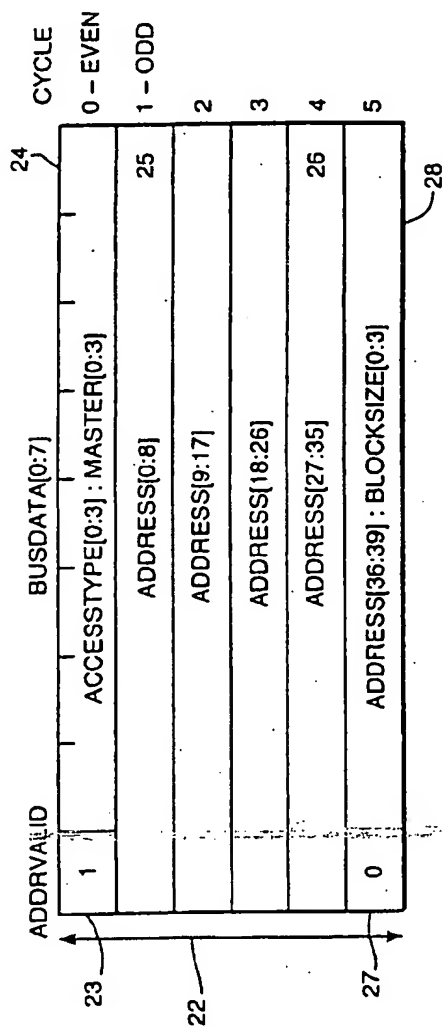
REJECT (NACK) CONTROL PACKET

	ADDRVALID	BUSDATA[0:7]	CYCLE
23	1	ACCESSTYPE[0:3] : MASTER[0:3] = 0	0 – EVEN
28	0	INFO[0:7]	1 – ODD

FIG_5

	ADDRVALID	BUSDATA[0:7]		CYCLE
23	1	ACCESSTYPE[0:3] : MASTER[0:3]	24	0
		ADDRESS[0:8]	25	1
22		ADDRESS[9:17]		2
		ADDRESS[18:26]		3
		ADDRESS[27:35]	26	4
27	1	ADDRESS[36:39] : BLOCKSIZE[0:3]	28	5
		XXX	29	6
	0	XXX	30	7
		XXX	31	8
	0	XXX	32	9
29	0	INVALID: REQUEST[1:7]	33	10
	0	REQUEST[8:15]	34	11

FIG_6

FIG_7A

FIG_7B

CLOCK1

CLOCK2

53

50

CLK

54

CHIP
N

52

CHIP
O

51

FIG_8A

55   56   57   58

CHIP N 52

55   56   57   58

CHIP O 51

59   60

FIG_8B

FIG_9

66

19

20

66

20

65

18

15 18

16 15

17

19

18

15 15

16 16

17 17

FIG__10

FIG_11

SAMPLE

83

84

68

69

SENSE

INPUT −

INPUT REFERENCE

INPUT +

PAD

FIG_12

FIG_13

BUS CLOCK1 53

BUS CLOCK2 54

DELAY LINE OUTPUT 107

DELAY LINE OUTPUT 108

INTERNAL CLOCK 73

INTERNAL CLOCK COMPLEMENT 74

INPUT SAMPLE

121

123

120

122

124

125

128

129

126

127

CLOCK

RESET IN

RESET OUT

BUS DATA [0:7]

N

N + 1

N + 2

FIG_ 14

FIG_15

1

METHOD OF OPERATING A MEMORY DEVICE HAVING A VARIABLE DATA INPUT LENGTH

This application is a continuation of application Ser. No. 09/252,997 now U.S. Pat. No. 6,034,918, which is a continuation of application Ser. No. 09/196,199, filed on Nov. 20, 1998 now U.S. Pat. No. 5,838,195, which is a continuation of application Ser. No. 08/798,520, filed on Feb. 10, 1997 (now U.S. Pat. No. 5,841,580); which is a division of application Ser. No. 08/448,657, filed May 24, 1995 (now U.S. Pat. No. 5,638,334); which is a division of application Ser. No. 08/222,646, filed on Mar. 31, 1994 (now U.S. Pat. No. 5,513,327); which is a continuation of application Ser. No. 07/954,945, filed on Sep. 30, 1992 (now U.S. Pat. No. 5,319,755); which is a continuation of application Ser. No. 07/510,898, filed on Apr. 18, 1990 (now abandoned).

FIELD OF THE INVENTION

An integrated circuit bus interface for computer and video systems is described which allows high speed transfer of blocks of data, particularly to and from memory devices, with reduced power consumption and increased system reliability. A new method of physically implementing the bus architecture is also described.

BACKGROUND OF THE INVENTION

Semiconductor computer memories have traditionally been designed and structured to use one memory device for each bit, or small group of bits, of any individual computer word, where the word size is governed by the choice of computer. Typical word sizes range from 4 to 64 bits. Each memory device typically is connected in parallel to a series of address lines and connected to one of a series of data lines. When the computer seeks to read from or write to a specific memory location, an address is put on the address lines and some or all of the memory devices are activated using a separate device select line for each needed device. One or more devices may be connected to each data line but typically only a small number of data lines are connected to a single memory device. Thus data line 0 in connected to device(s) 0; data line 1 is connected to device(s) 1, and so on.

Data is thus accessed or provided in parallel for each memory read or write operation. For the system to operate properly, every single memory bit in every memory device must operate dependably and correctly.

To understand the concept of the present invention, it is helpful to review the architecture of conventional memory devices. Internal to nearly all types of memory devices (including the most widely used Dynamic Random Access Memory (DRAM), Static RAM (SRAM) and Read Only Memory (ROM) devices), a large number of bits are accessed in parallel each time the system carries out a memory access cycle. However, only a small percentage of accessed bits which are available internally each time the memory device is cycled ever make it across the device boundary to the external world.

Referring to FIG. 1, all modern DRAM, SRAM and ROM designs have internal architectures with row (word) lines 5 and column (bit) lines 6 to allow the memory cells to tile a two dimensional area 1. One bit of data is stored at the intersection of each word and bit line. When a particular word line is enabled, all of the corresponding data bits are transferred onto the bit lines. Some prior art DRAMs take advantage of this organization to reduce the number of pins needed to transmit the address. The address of a given

2

memory cell Is split into two addresses, row and column, each of which can be Multiplexed over a bus only half as wide as the memory cell address of the prior art would have required.

COMPARISON WITH PRIOR ART

Prior art memory systems have attempted to solve the problem of high speed access to memory with limited success.

U.S. Pat. No. 3,821,715 (Hoff et. al.), was issued to Intel Corporation for the earliest 4-bit microprocessor. That patent describes a bus connecting a single central processing unit (CPU) with multiple RAMs and ROMs. That bus multiplexes addresses and data over a 4-bit wide bus and uses point-to-point control signals to select particular RAMs or ROMs. The access time is fixed and only a single processing element is permitted. There is no block-mode type of operation, and most important, not all of the interface signals between the devices are bused (the ROM and RAM control lines and the RAM select lines are point-to-point).

In U.S. Pat. No. 4,315,308 (Jackson), a bus connecting a single CPU to a bus interface unit is described.

The invention uses multiplexed address, data, and control information over a single 16-bit wide bus. Block-mode operations are defined, with the length of the block sent as part of the control sequence. In addition, variable access-time operations using a "stretch" cycle signal are provided. There are no multiple processing elements and no capability for multiple outstanding requests, and again, not all of the interface signals are bused.

In U.S. Pat. No. 4,449,207 (Kung, et. al.), a DRAM is described which multiplexes address and data on an internal bus. The external interface to this DRAM is conventional, with separate control, address and data connections.

In U.S. Pat. Nos. 4,764,846 and 4,706,166 (Go), a 3-D package arrangement of stacked die with connections along a single edge is described. Such packages are difficult to use because of the point-to-point wiring required to interconnect conventional memory devices with processing elements. Both patents describe complex schemes for solving these problems. No attempt is made to solve the problem by changing the interface.

In U.S. Pat. No. 3,969,706 (Proebsting, et. al.), the current state-of-the-art DRAM interface is described. The address is two-way multiplexed and there are separate pins for data and control (RAS, CAS, WE, CS). The number of pins grows with the size of the DRAM, and many of the connections must be made point-to-point in a memory system using such DRAMs.

There are many backplane buses described in the prior art, but not in the combination described or having the features of this invention. Many backplane buses multiplex addresses and data on a single bus (e.g., the NU bus). ELXSI and others have implemented split-transaction buses (U.S. Pat. Nos. 4,595,923 and 4,481,625 (Roberts)). ELXSI has also implemented a relatively low-voltage-swing current-mode ECL driver (approximately 1 V Swing). Address-space registers are implemented on most backplane buses, as is some form of block mode operation.

Nearly all modern backplane buses implement some type of arbitration scheme, but the arbitration scheme used in this invention differs from each of these. U.S. Pat. No. 4,837,682 (Culler), U.S. Pat. No. 4,818,985 (Ikeda), U.S. Pat. No. 4,779,089 (Theus) and U.S. Pat. No. 4,745,548 (Blahut) describe prior art schemes. All involve either log N extra

3

signals, (Theus, Blahut), where N is the number of potential bus requestors, or additional delay to get control of the bus (Ikeda, Culler). None of the buses described in patents or other literature use only bused connections. All contain some point-to-point connections on the backplane. None of the other aspects of this invention such as power reduction by fetching each data block from a single device or compact and low-cost 3-D packaging even apply to backplane buses.

The clocking scheme used in this invention has not been used before and in fact would be difficult to implement in backplane buses due to the signal degradation caused by connector stubs. U.S. Pat. No. 4,247,917 (Heller) describes a clocking scheme using two clock lines, but relies on ramp-shaped clock signals in contrast to the normal rise-time signals used in the present invention.

In U.S. Pat. No. 4,646,270 (Voss), a video RAM is described which implements a parallel-load, serial-out shift register on the output of a DRAM. This generally allows greatly improved bandwidth (and has been extended to 2, 4 and greater width shift-out paths.) The rest of the interfaces to the DRAM (RAS, CAS, multiplexed address, etc.) remain the same as for conventional DRAMS.

One object of the present invention is to use a new bus interface built into semiconductor devices to support high-speed access to large blocks of data from a single memory device by an external user of the data, such as a microprocessor, in an efficient and cost-effective manner.

Another object of this invention is to provide a clocking scheme to permit high speed clock signals to be sent along the bus with minimal clock skew between devices.

Another object of this invention is to allow mapping out defective memory devices or portions of memory devices.

Another object of this invention is to provide a method for distinguishing otherwise identical devices by assigning a unique identifier to each device.

Yet another object of this invention is to provide a method for transferring address, data and control information over a relatively narrow bus and to provide a method of bus arbitration when multiple devices seek to use the bus simultaneously.

Another object of this invention is to provide a method of distributing a high-speed memory cache within the DRAM chips of a memory system which is much more effective than previous cache methods.

Another object of this invention is to provide devices, especially DRAMs, suitable for use with the bus architecture of the invention.

SUMMARY OF INVENTION

The present invention includes a memory subsystem comprising at least two semiconductor devices, including at least one memory device, connected in parallel to a bus, where the bus includes a plurality of bus lines for carrying substantially all address, data and control information needed by said memory devices, where the control information includes device-select information and the bus has substantially fewer bus lines than the number of bits in a single address, and the bus carries device-select information without the need for separate device-select lines connected directly to individual devices.

Referring to FIG. 2, a standard DRAM 13, 14, ROM (or SRAM) 12, microprocessor CPU 11, I/O device, disk controller or other special purpose device such as a high speed switch is modified to use a wholly bus-based interface rather than the prior art combination of point-to-point and bus-

4

based wiring used with conventional versions of these devices. The new bus includes clock signals, power and multiplexed address, data and control signals. In a preferred implementation, 8 bus data lines and an AddressValid bus line carry address, data and control information for memory addresses up to 40 bits wide. Persons skilled in the art will recognize that 16 bus data lines or other numbers of bus data lines can be used to implement the teaching of this invention. The new bus is used to connect elements such as memory, peripheral, switch and processing units.

In the system of this invention, DRAMs and other devices receive address and control information over the bus and transmit or receive requested data over the same bus. Each memory device contains only a single bus interface with no other signal pins. Other devices that may be included in the system can connect to the bus and other non-bus lines, such as input/output lines. The bus supports large data block transfers and split transactions to allow a user to achieve high bus utilization. This ability to rapidly read or write a large block of data to one single device at a time is an important advantage of this invention.

The DRAMs that connect to this bus differ from conventional DRAMs in a number of ways. Registers are provided which may store control information, device identification, device-type and other information appropriate for the chip such as the address range for each independent portion of the device. New bus interface circuits must be added and the internals of prior art DRAM devices need to be modified so they can provide and accept data to and from the bus at the peak data rate of the bus. This requires changes to the column access circuitry in the DRAM, with only a minimal increase in die size. A circuit is provided to generate a low skew internal device clock for devices on the bus, and other circuits provide for demultiplexing input and multiplexing output signals.

High bus bandwidth is achieved by running the bus at a very high clock rate (hundreds of MHz). This high clock rate is made possible by the constrained environment of the bus. The bus lines are controlled-impedance, doubly-terminated lines. For a data rate of 500 MHz, the maximum bus propagation time is less than 1 ns (the physical bus length is about 10 cm). In addition, because of the packaging used, the pitch of the pins can be very close to the pitch of the pads. The loading on the bus resulting from the individual devices is very small. In a preferred implementation, this generally allows stub capacitances of 1–2 pf and inductances of 0.5–2 nH. Each device 15, 16, 17, shown in FIG. 3, only has pins on one side and these pins connect directly to the bus 18. A transceiver device 19 can be included to interface multiple units to a higher order bus through pins 20.

A primary result of the architecture of this invention is to increase the bandwidth of DRAM access. The invention also reduces manufacturing and production costs, power consumption, and increases packing density and system reliability.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a diagram which illustrates the basic 2-D organization of memory devices.

FIG. 2 is a schematic block diagram which illustrates the parallel connection of all bus lines and the serial Reset line to each device in the system.

FIG. 3 is a perspective view of a system of the invention which illustrates the 3-D packaging of semiconductor devices on the primary bus.

FIG. 4 shows the format of a request packet.

5

FIG. 5 shows the format of a retry response from a slave.

FIG. 6 shows the bus cycles after a request packet collision occurs on the bus and how arbitration is handled.

FIGS. 7a and 7b show the timing whereby signals from two devices can overlap temporarily and drive the bus at the same time.

FIGS. 8a and 8b show the connection and timing between bus clocks and devices on the bus.

FIG. 9 is a perspective view showing how transceivers can be used to connect a number of bus units to a transceiver bus.

FIG. 10 is a block and schematic diagram of input/output circuitry used to connect devices to the bus.

FIG. 11 is a schematic diagram of a clocked sense-amplifier used as a bus input receiver.

FIG. 12 is a block diagram showing how the internal device clock is generated from two bus clock signals using a set of adjustable delay lines.

FIG. 13 is a timing diagram showing the relationship of signals in the block diagram of FIG. 12.

FIG. 14 is a timing diagram of a preferred means of implementing the reset procedure of this invention.

FIG. 15 is a diagram illustrating the general organization of a 4 Mbit DRAM divided into 8 subarrays.

DETAILED DESCRIPTION

The present invention is designed to provide a high speed, multiplexed bus for communication between processing devices and memory devices and to provide devices adapted for use in the bus system. The invention can also be used to connect processing devices and other devices, such as I/O interfaces or disk controllers, with or without memory devices on the bus. The bus consists of a relatively small number of lines connected in parallel to each device on the bus. The bus carries substantially all address, data and control information needed by devices for communication with other devices on the bus. In many systems using the present invention, the bus carries almost every signal between every device in the entire system. There is no need for separate device-select lines since device-select information for each device on the bus is carried over the bus. There is no need for separate address and data lines because address and data information can be sent over the same lines. Using the organization described herein, very large addresses (40 bits in the preferred implementation) and large data blocks (1024 bytes) can be sent over a small number of bus lines (8 plus one control line in the preferred implementation).

Virtually all of the signals needed by a computer system can be sent over the bus. Persons skilled in the art recognize that certain devices, such as CPUs, may be connected to other signal lines and possibly to independent buses, for example a bus to an independent cache memory, in addition to the bus of this invention. Certain devices, for example cross-point switches, could be connected to multiple, independent buses of this invention. In the preferred implementation, memory devices are provided that have no connections other than the bus connections described herein and CPUs are provided that use the bus of this invention as the principal, if not exclusive, connection to memory and to other devices on the bus.

All modern DRAM, SRAM and ROM designs have internal architectures with row (word) and column (bit) lines to efficiently tile a 2-D area. Referring to FIG. 1, one bit of data is stored at the intersection of each word line 5 and bit

6

line 6. When a particular word line is enabled, all of the corresponding data bits are transferred onto the bit lines. This data, about 4000 bits at a time in a 4 MBit DRAM, is then loaded into column sense amplifiers 3 and held for use by the I/O circuits.

In the invention presented here, the data from the sense amplifiers is enabled 32 bits at a time onto an internal device bus running at approximately 125 MHz. This internal device bus moves the data to the periphery of the devices where the data is multiplexed into an 8-bit wide external bus interface, running at approximately 500 MHz.

The bus architecture of this invention connects master or bus controller devices, such as CPUs, Direct Memory Access devices (DMAs) or Floating Point Units (FPUs), and slave devices, such as DRAM, SRAM or ROM memory devices. A slave device responds to control signals; a master sends control signals. Persons skilled in the art realize that some devices may behave as both master and slave at various times, depending on the mode of operation and the state of the system. For example, a memory device will typically have only slave functions, while a DMA controller, disk controller or CPU may include both slave and master functions. Many other semiconductor devices, including I/O devices, disk controllers, or other special purpose devices such as high speed switches can be modified for use with the bus of this invention.

Each semiconductor device contains a set of internal registers, preferably including a device identification (device ID) register, a device-type descriptor register, control registers and other registers containing other information relevant to that type of device. In a preferred implementation, semiconductor devices connected to the bus contain registers which specify the memory addresses contained within that device and access-time registers which store a set of one or more delay times at which the device can or should be available to send or receive data.

Most of these registers can be modified and preferably are set as part of an initialization sequence that occurs when the system is powered up or reset. During the initialization sequence each device on the bus is assigned a unique device ID number, which is stored in the device ID register. A bus master can then use these device ID numbers to access and set appropriate registers in other devices, including access-time registers, control registers, and memory registers, to configure the system. Each slave may have one or several access-time registers (four in a preferred embodiment). In a preferred embodiment, one access-time register in each slave is permanently or semi-permanently programmed with a fixed value to facilitate certain control functions. A preferred implementation of an initialization sequence is described below in more detail.

All information sent between master devices and slave devices is sent over the external bus, which, for example, may be 8 bits wide. This is accomplished by defining a protocol whereby a master device, such as a microprocessor, seizes exclusive control of the external bus (i.e., becomes the bus master) and initiates a bus transaction by sending a request packet (a sequence of bytes comprising address and control information) to one or more slave devices on the bus. An address can consist of 16 to 40 or more bits according to the teachings of this invention. Each slave on the bus must decode the request packet to see if that slave needs to respond to the packet. The slave that the packet is directed to must then begin any internal processes needed to carry out the requested bus transaction at the requested time. The requesting master may also need to transact certain internal

processes before the bus transaction begins. After a specified access time the slave(s) respond by returning one or more bytes (8 bits) of data or by storing information made available from the bus. More than one access time can be provided to allow different types of responses to occur at different times.

A request packet and the corresponding bus access are separated by a selected number of bus cycles, allowing the bus to be used in the intervening bus cycles by the same or other masters for additional requests or brief bus accesses. Thus multiple, independent accesses are permitted, allowing maximum utilization of the bus for transfer of short blocks of data. Transfers of long blocks of data use the bus efficiently even without overlap because the overhead due to bus address, control and access times is small compared to the total time to request and transfer the block.

Device Address Mapping

Another unique aspect of this invention is that each memory device is a complete, independent memory subsystem with all the functionality of a prior art memory board in a conventional backplane-bus computer system. Individual memory devices may contain a single memory section or may be subdivided into more than one discrete memory section. Memory devices preferably include memory address registers for each discrete memory section. A failed memory device (or even a subsection of a device) can be "mapped out" with only the loss of a small fraction of the memory, maintaining essentially full system capability. Mapping out bad devices can be accomplished in two ways, both compatible with this invention.

The preferred method uses address registers in each memory device (or independent discrete portion thereof) to store information which defines the range of bus addresses to which this memory device will respond. This is similar to prior art schemes used in memory boards in conventional backplane bus systems. The address registers can include a single pointer, usually pointing to a block of known size, a pointer and a fixed or variable block size value or two pointers, one pointing to the beginning and one to the end (or to the "top" and "bottom") of each memory block. By appropriate settings of the address registers, a series of functional memory devices or discrete memory sections can be made to respond to a contiguous range of addresses, giving the system access to a contiguous block of good memory, limited primarily by the number of good devices connected to the bus. A block of memory in a first memory device or memory section can be assigned a certain range of addresses, then a block of memory in a next memory device or memory section can be assigned addresses starting with an address one higher (or lower, depending on the memory structure) than the last address of the previous block.

Preferred devices for use in this invention include device-type register information specifying the type of chip, including how much memory is available in what configuration on that device. A master can perform an appropriate memory test, such as reading and writing each memory cell in one or more selected orders, to test proper functioning of each accessible discrete portion of memory (based in part on information like device ID number and device-type) and write address values (up to 40 bits in the preferred embodiment, 10^{12} bytes), preferably contiguous, into device address-space registers. Non-functional or impaired memory sections can be assigned a special address value which the system can interpret to avoid using that memory.

The second approach puts the burden of avoiding the bad devices on the system master or masters. CPUs and DMA controllers typically have some sort of translation look-aside

buffers (TLBs) which map virtual to physical (bus) addresses. With relatively simple software, the TLBs can be programmed to use only working memory (data structures describing functional memories are easily generated). For masters which don't contain TLBs (for example, a video display generator), a small, simple RAM can be used to map a contiguous range of addresses onto the addresses of the functional memory devices.

Either scheme works and permits a system to have a significant percentage of non-functional devices and still continue to operate with the memory which remains. This means that systems built with this invention will have much improved reliability over existing systems, including the ability to build systems with almost no field failures.

Bus

The preferred bus architecture of this invention comprises 11 signals: BusData[0:7]; AddrValid; Clk1 and Clk2; plus an input reference level and power and ground lines connected in parallel to each device. Signals are driven onto the bus during conventional bus cycles. The notation "Signal[i:j]" refers to a specific range of signals or lines, for examples, BusData[0:7] means BusData0, BusData1, . . . , BusData7. The bus lines for BusData[0:7] signals form a byte-wide, multiplexed data/address/control bus. AddrValid is used to indicate when the bus is holding a valid address request, and instructs a slave to decode the bus data as an address and, if the address is included on that slave, to handle the pending request. The two clocks together provide a synchronized, high speed clock for all the devices on the bus. In addition to the bused signals, there is one other line (ResetIn, ResetOut) connecting each device in series for use during initialization to assign every device in the system a unique device ID number (described below in detail).

To facilitate the extremely high data rate of this external bus relative to the gate delays of the internal logic, the bus cycles are grouped into pairs of even/odd cycles. Note that all devices connected to a bus should preferably use the same even/odd labeling of bus cycles and preferably should begin operations on even cycles. This is enforced by the clocking scheme.

Protocol and Bus Operation

The bus uses a relatively simple, synchronous, split-transaction, block-oriented protocol for bus transactions. One of the goals of the system is to keep the intelligence concentrated in the masters, thus keeping the slaves as simple as possible (since there are typically many more slaves than masters). To reduce the complexity of the slaves, a slave should preferably respond to a request in a specified time, sufficient to allow the slave to begin or possibly complete a device-internal phase including any internal actions that must precede the subsequent bus access phase. The time for this bus access phase is known to all devices on the bus—each master being responsible for making sure that the bus will be free when the bus access begins. Thus the slaves never worry about arbitrating for the bus. This approach eliminates arbitration in single master systems, and also makes the slave-bus interface simpler.

In a preferred implementation of the invention, to initiate a bus transfer over the bus, a master sends out a request packet, a contiguous series of bytes containing address and control information. It is preferable to use a request packet containing an even number of bytes and also preferable to start each packet on an even bus cycle.

The device-select function is handled using the bus data lines. AddrValid is driven, which instructs all slaves to decode the request packet address, determine whether they contain the requested address, and if they do, provide the

data back to the master (in the case of a read request) or accept data from the master (in the case of a write request) in a data block transfer. A master can also select a specific device by transmitting a device ID number in a request packet. In a preferred implementation, a special device ID number is chosen to indicate that the packet should be interpreted by all devices on the bus. This allows a master to broadcast a message, for example to set a selected control register of all devices with the same value.

The data block transfer occurs later at a time specified in the request packet control information, preferably beginning on an even cycle. A device begins a data block transfer almost immediately with a device-internal phase as the device initiates certain functions, such as setting up memory addressing, before the bus access phase begins. The time after which a data block is driven onto the bus lines is selected from values stored in slave access-time registers. The timing of data for reads and writes is preferably the same; the only difference is which device drives the bus. For reads, the slave drives the bus and the master latches the values from the bus. For writes the master drives the bus and the selected slave latches the values from the bus.

In a preferred implementation of this invention shown in FIG. 4, a request packet 22 contains 6 bytes of data—4.5 address bytes and 1.5 control bytes. Each request packet uses all nine bits of the multiplexed data/address lines (AddrValid 23+BusData[0:7] 24) for all six bytes of the request packet. setting 23 AddrValid=1 in an otherwise unused even cycle indicates the start of an request packet (control information).

In a valid request packet, AddrValid 27 must be 0 in the last byte. Asserting this signal in the last byte invalidates the request packet. This is used for the collision detection and arbitration logic (described below). Bytes 25–26 contain the first 35 address bits, Address[0:35]. The last byte contains AddrValid 27 (the invalidation switch) and 28, the remaining address bits, Address[36:39], and BlockSize[0:3] (control information).

The first byte contains two 4 bit fields containing control information, AccessType[0:3], an op code (operation code) which, for example, specifies the type of access, and Master [0:3], a position reserved for the master sending the packet to include its master ID number. Only master numbers 1 through 15 are allowed—master number 0 is reserved for special system commands. Any packet with Master[0:3]=0 is an invalid or special packet and is treated accordingly.

The AccessType field specifies whether the requested operation is a read or write and the type of access, for example, whether it is to the control registers or other parts of the device, such as memory. In a preferred implementation, AccessType[0] is a Read/Write switch: if it is a 1, then the operation calls for a read from the slave (the slave to read the requested memory block and drive the memory contents onto the bus); if it is a 0, the operation calls for a write into the slave (the slave to read data from the bus and write it to memory). AccessType[1:3] provides up to 8 different access types for a slave. AccessType[1:2] preferably indicates the timing of the response, which is stored in an access-time register, AccessRegN. The choice of access-time register can be selected directly by having a certain op code select that register, or indirectly by having a slave respond to selected op codes with pre-selected access times (see table below). The remaining bit, AccessType[3] may be used to send additional information about the request to the slaves.

One special type of access is control register access, which involves addressing a selected register in a selected

slave. In the preferred implementation of this invention, AccessType[1:3] equal to zero indicates a control register request and the address field of the packet indicates the desired control register. For example, the most significant two bytes can be the device ID number (specifying which slave in being addressed) and the least significant three bytes can specify a register address and may also represent or include data to be loaded into that control register. Control register accesses are used to initialize the access-time registers, so it is preferable to use a fixed response time which can be preprogrammed or even hard wired, for example the value in AccessReg0, preferably 8 cycles. Control register access can also be used to initialize or modify other registers, including address registers.

The method of this invention provides for access mode control specifically for the DRAMs. One such access mode determines whether the access is page mode or normal RAS access. In normal mode (in conventional DRAMS and in this invention), the DRAM column sense amps or latches have been precharged to a value intermediate between logical 0 and 1. This precharging allows access to a row in the R to begin as soon as the access request for either inputs (writes) or outputs (reads) is received and allows the column sense amps to sense data quickly. In page mode (both conventional and in this invention), the DRAM holds the data in the column sense amps or latches from the previous read or write operation. If a subsequent request to access data is directed to the same row, the DRAM does not need to wait for the data to be sensed (it has been sensed already) and access time for this data is much shorter than the normal access time. Page mode generally allows much faster access to data but to a smaller block of data (equal to the number of sense amps). However, if the requested data is not in the selected row, the access time is longer than the normal access time, since the request must wait for the RAM to precharge before the normal mode access can start. Two access-time registers in each DRAM preferably contain the access times to be used for normal and for page-mode accesses, respectively.

The access mode also determines whether the DRAM should precharge the sense amplifiers or should save the contents of the sense amps for a subsequent page mode access. Typical settings are "precharge after normal access" and "save after page mode access" but "precharge after page mode access" or "save after normal access" are allowed, selectable modes of operation. The DRAM can also be set to precharge the sense amps if they are not accessed for a selected period of time.

In page mode, the data stored in the DRAM sense amplifiers may be accessed within much less time than it takes to read out data in normal mode (~10–20 nS vs. 40–100 nS). This data may be kept available for long periods. However, if these sense amps (and hence bit lines) are not precharged after an access, a subsequent access to a different memory word (row) will suffer a precharge time penalty of about 40–100 nS because the sense amps must precharge before latching in a new value.

The contents of the sense amps thus may be held and used as a cache, allowing faster, repetitive access to small blocks of data. DRAM-based page-mode caches have been attempted in the prior art using conventional DRAM organizations but they are not very effective because several chips are required per computer word. Such a conventional page-mode cache contains many bits (for example, 32 chips×4 Kbits) but has very few independent storage entries. In other words, at any given point in time the sense amps hold only a few different blocks or memory "locales" (a

single block of 4K words, in the example above). Simulations have shown that upwards of 100 blocks are required to achieve high hit rates (>90% of requests find the requested data already in cache memory) regardless of the size of each block. See, for example, Anant Agarwal, et. al., "An Analytic Cache Model," *ACM Transactions on Computer Systems*, Vol. 7(2), pp. 184–215 (May 1989).

The organization of memory in the present invention allows each DRAM to hold one or more (4 for 4 MBit DRAMS) separately-addressed and independent blocks of data. A personal computer or workstation with 100 such DRAMs (i.e. 400 blocks or locales) can achieve extremely high, very repeatable hit rates (98–99% on average) as compared to the lower (50–80%), widely varying bit rates using DRAMS organized in the conventional fashion. Further, because of the time penalty associated with the deferred precharge on a "miss" of the page-mode cache, the conventional DRAM-based page-mode cache generally has been found to work less well than no cache at all.

For DRAM slave access, the access types are preferably used in the following way:

AccessType[1:3]	Use	AccessTime
0	Control Register Access	Fixed, 8[AccessReg0]
1	Unused	Fixed, 8[AccessReg0]
2–3	Unused	AccessReg1
4–5	Page Mode DRAM access	AccessReg2
6–7	Normal DRAM access	AccessReg3

Persons skilled in the art will recognize that a series of available bits could be designated as switches for controlling these access modes. For example:

AccessType[2] page mode/normal switch

AccessType[3] precharge/save-data switch

BlockSize[0:3] specifies the site of the data block transfer. If BlockSize[0] is 0, the remaining bits are the binary representation of the block size (0–7). If BlockSize[0] is 1, then the remaining bits give the block size as a binary power of 2, from 8 to 1024. A zero-length block can be interpreted as a special command, for example, to refresh a DRAM without returning any data, or to change the DRAM from page mode to normal access mode or vice-versa.

BlockSize[0:2]	Number of Bytes in Block
0–7	0–7 respectively
8	8
9	16
10	32
11	64
12	128
13	256
14	512
15	1024

Persons skilled in the art will recognize that other block size encoding schemes or values can be used.

In most cases, a slave will respond at the selected access time by reading or writing data from or to the bus over bus lines BusData[0:7] and AddrValid will be at logical 0. In a preferred embodiment, substantially each memory access will involve only a single memory device, that is, a single block will be read from or written to a single memory device.

Retry Format

In some cases, a slave may not be able to respond correctly to a request, e.g., for a read or write. In such a situation, the slave should return an error message, sometimes called a N(o)ACK(nowledge) or retry message. The retry message can include information about the condition requiring a retry, but this increases system requirements for circuitry in both slave and masters. A simple message indicating only that an error has occurred allows for a less complex slave, and the master can take whatever action is needed to understand and correct the cause of the error.

For example, under certain conditions a slave might not be able to supply the requested data. During a page-mode access, the DRAM selected must be in page mode and the requested address must match the address of the data held in the sense amps or latches. Each DRAM can check for this match during a page-mode access. If no match is found, the DRAM begins precharging and returns a retry message to the master during the first cycle of the data block (the rest of the returned block is ignored). The master then must wait for the precharge time (which in set to accommodate the type of slave in question, stored in a special register, PreChargeReg), and then resend the request as a normal DRAM access (AccessType=6 or 7).

In the preferred form of the present invention, a slave signals a retry by driving AddrValid true at the time the slave was supposed to begin reading or writing data. A master which expected to write to that slave must monitor AddrValid during the write and take corrective action if it detects a retry message. FIG. 5 illustrates the format of a retry message 28 which is useful for read requests, consisting of 23 AddrValid=1 with Master[0:3]=0 in the first (even) cycle. Note that AddrValid is normally 0 for data block transfers and that there is no master 0 (only 1 through 15 are allowed). All DRAMs and masters can easily recognize such a packet as an invalid request packet, and therefore a retry message. In this type of bus transaction all of the fields except for Master[0:3] and AddrValid 23 may be used an information fields, although in the implementation described, the contents are undefined. Persons skilled in the art recognize that another method of signifying a retry message is to add a DataInvalid line and signal to the bus. This signal could be asserted in the case of a NACK.

Bus Arbitration

In the case of a single master, there are by definition no arbitration problems. The master sends request packets and keeps track of periods when the bus will be busy in response to that packet. The master can schedule multiple requests so that the corresponding data block transfers do not overlap.

The bus architecture of this invention is also useful in configurations with multiple masters. When two or more masters are on the same bus, each master must keep track of all the pending transactions, so each master knows when it can send a request packet and access the corresponding data block transfer. Situations will arise, however, where two or more masters send a request packet at about the same time and the multiple requests must be detected, then sorted out by some sort of bus arbitration.

There are many ways for each master to keep track of when the bus is and will be busy. A simple method is for each master to maintain a bus-busy data structure, for example by maintaining two pointers, one to indicate the earliest point in the future when the bus will be busy and the other to indicate the earliest point in the future when the bus will be free, that is, the end of the latest pending data block transfer. Using this information, each master can determine whether and when there is enough time to send a request packet (as

13 14

described above under Protocol) before the bus becomes busy with another data block transfer and whether the corresponding data block transfer will interfere with pending bus transactions. Thus each master must read every request packet and update its bus-busy data structure to maintain information about when the bus is and will be free.

With two or more masters on the bus, masters will occasionally transmit independent request packets during the same bus cycle. Those multiple requests will collide as each such master drives the bus simultaneously with different information, resulting in scrambled request information and neither desired data block transfer. In a preferred form of the invention, each device on the bus seeking to write a logical 1 on a BusData or AddrValid line drives that line with a current sufficient to sustain a voltage greater than or equal to the high-logic value for the system. Devices do not drive lines that should have a logical 0; those lines are simply held at a voltage corresponding to a low-logic value. Each master tests the voltage on at least some, preferably all, bus data and the AddrValid lines so the master can detect a logical '1' where the expected level is '0' on a line that it does not drive during a given bus cycle but another master does drive.

Another way to detect collisions is to select one or more bus lines for collision signalling. Each master sending a request drives that line or lines and monitors the selected lines for more than the normal drive current (or a logical value of ">1"), indicating requests by more than one master. Persons skilled in the art will recognize that this can be implemented with a protocol involving BusData and AddrValid lines or could be implemented using an additional bus line.

In the preferred form of this invention, each master detects collisions by monitoring lines which it does not drive to see if another master is driving those lines. "Referring to FIG. 4, the first byte of the request packet includes the number of each master attempting to use the bus (Master [0:3]). If two masters send packet requests starting at the same point in time, the master numbers will be logical "or"ed together by at least those masters, and thus one or both of the masters, by monitoring the data on the bus and comparing what it sent, can detect a collision. For instance if requests by masters number 2 (0010) and 5 (0101) collide, the bus will be driven with the value Master[0:3]=7 (0010+ 0101=0111). Master number 5 will detect that the signal Master[2]=1 and master 2 will detect that Master[1] and Master[3]=1, telling both masters that a collision has occurred. Another example in masters 2 and 11, for which the bus will be driven with the value Master[0:3]=11 (0010+ 1011=1011), and although master 11 can't readily detect this collision, master 2 can. When any collision is detected, each master detecting a collision drives the value of AddrValid 27 in byte 5 of the request packet 22 to 1, which is detected by all masters, including master 11 in the second example above, and forces a bus arbitration cycle, described below.

Another collision condition may arise where master A sends a request packet in cycle 0 and master B tries to send a request packet starting in cycle 2 of the first request packet, thereby overlapping the first request packet. This will occur from time to time because the bus operates at high speeds, thus the logic in a second-initiating master may not be fast enough to detect a request initiated by a first master in cycle 0 and to react fast enough by delaying its own request. Master B eventually notices that it wasn't supposed to try to send a request packet (and consequently almost surely destroyed the address that master A was trying to send), and, as in the example above of a simultaneous collision, drives a 1 on AddrValid during byte 5 of the first request packet 27

forcing an arbitration. The logic in the preferred implementation is fast enough that a master should detect a request packet by another master by cycle 3 of the first request packet, so no master is likely to attempt to send a potentially colliding request packet later than cycle 2.

Slave devices do not need to detect a collision directly, but they must wait to do anything irrecoverable until the last byte (byte 5) is read to ensure that the packet is valid. A request packet with Master[0:3] equal to 0 (a retry signal) is ignored and does not cause a collision. The subsequent bytes of such a packet are ignored.

To begin arbitration after a collision, the masters wait a preselected number of cycles after the aborted request packet (4 cycles in a preferred implementation), then use the next free cycle to arbitrate for the bus (the next available even cycle in the preferred implementation). Each colliding master signals to all other colliding masters that it seeks to send a request packet, a priority is assigned to each of the colliding masters, then each master is allowed to make its request in the order of that priority.

FIG. 6 illustrates one preferred way of implementing this arbitration. Each colliding master signals its intent to send a request packet by driving a single BusData line during a single bus cycle corresponding to its assigned master number (1–15 in the present example). During two-byte arbitration cycle 29, byte 0 is allocated to requests 1–7 from masters 1–7, respectively, (bit 0 is not used) and byte 1 is allocated to requests 8–15 from masters 8–15, respectively. At least one device and preferably each colliding master reads the values on the bus during the arbitration cycles to determine and store which masters desire to use the bus. Persons skilled in the art will recognize that a single byte can be allocated for arbitration requests if the system includes more bus lines than masters. More than 15 masters can be accommodated by using additional bus cycles.

A fixed priority scheme (preferably using the master numbers, selecting lowest numbers first) is then used to prioritize, then sequence the requests in a bus arbitration queue which is maintained by at least one device. These requests are queued by each master in the bus-busy data structure and no further requests are allowed until the bus arbitration queue is cleared. Persons skilled in the art will recognize that other priority schemes can be used, including assigning priority according to the physical location of each master.

System Configuration/Reset

In the bus-based system of this invention, a mechanism is provided to give each device on the bus a unique device identifier (device ID) after power-up or under other conditions as desired or needed by the system. A master can then use this device ID to access a specific device, particularly to set or modify registers of the specified device, including the control and address registers. In the preferred embodiment, one master is assigned to carry out the entire system configuration process. The master provides a series of unique device ID numbers for each unique device connected to the bus system. In the preferred embodiment, each device connected to the bus contains a special device-type register which specifies the type of device, for instance CPU, 4 KBit memory, 64 MBit memory or disk controller. The configuration master should check each device, determine the device type and set appropriate control registers, including access-time registers. The configuration master should check each memory device and set all appropriate memory address registers.

One means to set up unique device ID numbers in to have each device to select a device ID in sequence and store the

value in an internal device ID register. For example, a master can pass sequential device ID numbers through shift registers in each of a series of devices, or pass a token from device to device whereby the device with the token reads in device ID information from another line or lines. In a preferred embodiment, device ID numbers are assigned to devices according to their physical relationship, for instance, their order along the bus.

In a preferred embodiment of this invention, the device ID setting is accomplished using a pair of pins on each device, ResetIn and ResetOut. These pins handle normal logic signals and are used only during device ID configuration. On each rising edge of the clock, each device copies ResetIn (an input) into a four-stage reset shift register. The output of the reset shift register is connected to ResetOut, which in turn connects to ResetIn for the next sequentially connected device. Substantially all devices on the bus are thereby daisy-chained together. A first reset signal, for example, while ResetIn at a device is a logical 1, or when a selected bit of the reset shift register goes from zero to non-zero, causes the device to hard reset, for example by clearing all internal registers and resetting all state machines. A second reset signal, for example, the falling edge of ResetIn combined with changeable values on the external bus, causes that device to latch the contents of the external bus into the internal device ID register (Device[0:7]).

To reset all devices on a bus, a master sets the ResetIn line of the first device to a "1" for long enough to ensure that all devices on the bus have been reset (4 cycles times the number of devices—note that the maximum number of devices on the preferred bus configuration is 256 (8 bits), so that 1024 cycles is always enough time to reset all devices.) Then ResetIn is dropped to "0" and the BusData lines are driven with the first followed by successive device ID numbers, changing after every 4 clock pulses. Successive devices set those device ID numbers into the corresponding device ID register on the falling edge of ResetIn propagates through the shift registers of the daisy-chained devices. FIG. 14 shows ResetIn at a first device going low while a master drives a first device ID onto the bus data lines BusData[0:3]. The first device then latches in that first device ID. After four clock cycles, the master changes BusData[0:3] to the next device ID number and ResetOut at the first device goes low, which pulls ResetIn for the next daisy-chained device low, allowing the next device to latch in the next device ID number from BusData[0:3]. In the preferred embodiment, one master is assigned device ID 0 and it is the responsibility of that master to control the ResetIn line and to drive successive device ID numbers onto the bus at the appropriate times. In the preferred embodiment, each device waits two clock cycles after ResetIn goes low before latching in a device ID number from BusData[0:3].

Persons skilled in the art recognize that longer device ID numbers could be distributed to devices by having each device read in multiple bytes from the bus and latch the values into the device ID register. Persons skilled in the art also recognize that there are alternative ways of getting device ID numbers to unique devices. For instance, a series of sequential numbers could be clocked along the ResetIn line and at a certain time each device could be instructed to latch the current reset shift register value into the device ID register.

The configuration master should choose and set an access time in each access-time register in each slave to a period sufficiently long to allow the slave to perform an actual, desired memory access. For example, for a normal DRAM access, this time must be longer than the row address strobe

(RAS) access time. If this condition is not met, the slave may not deliver the correct data. The value stored in a slave access-time register is preferably one-half the number of bus cycles for which the slave device should wait before using the bus in response to a request. Thus an access time value of '1' would indicate that the slave should not access the bus until at least two cycles after the last byte of the request packet has been received. The value of AccessReg0 is preferably fixed at 8 (cycles) to facilitate access to control registers.

The bus architecture of this invention can include more than one master device. The reset or initialization sequence should also include a determination of whether there are multiple masters on the bus, and if so to assign unique master ID numbers to each. Persons skilled in the art will recognize that there are many ways of doing this. For instance, the master could poll each device to determine what type of device it is, for example, by reading a special register then, for each master device, write the next available master ID number into a special register.

Error detection and correction ("ECC") methods well known in the art can be implemented in this system. ECC information typically is calculated for a block of data at the time that block of data is first written into memory. The data block usually has an integral binary size, e.g. 256 bits, and the ECC information uses significantly fewer bits. A potential problem arises in that each binary data block in prior art schemes typically is stored with the ECC bits appended, resulting in a block size that is not an integral binary power.

In a preferred embodiment of this invention, ECC information is stored separately from the corresponding data, which can then be stored in blocks having integral binary size. ECC information and corresponding data can be stored, for example, in separate DRAM devices. Data can be read without ECC using a single request packet, but to write or read error-corrected data requires two request packets, one for the data and a second for the corresponding ECC information. ECC information may not always be stored permanently and in some situations the ECC information may be available without sending a request packet or without a bus data block transfer.

In a preferred embodiment, a standard data block size can be selected for use with ECC, and the ECC method will determine the required number of bits of information in a corresponding ECC block. RAMs containing ECC information can be programmed to store an access time that in equal to: (1) the access time of the normal RAM (containing data) plus the time to access a standard data block (for corrected data) minus the time to send a request packet (6 bytes); or (2) the access time of a normal RAM minus the time to access a standard ECC block minus the time to send a request packet. To read a data block and the corresponding ECC block, the master simply issues a request for the data immediately followed by a request for the ECC block. The ECC RAM will wait for the selected access time then drive its data onto the bus right after (in case (1) above)) the data RAM has finished driving out the data block. Persons skilled in the art will recognize that the access time described in case (2) above can be used to drive ECC data before the data is driven onto the bus lines and will recognize that writing data can be done by analogy with the method described for a read. Persons skilled in the art will also recognize the adjustments that must be made in the bus-busy structure and the request packet arbitration methods of this invention in order to accommodate these paired ECC requests.

Since this system is quite flexible, the system designer can choose the size of the data blocks and the number of ECC

17   18

bits using the memory devices of this invention. Note that the data stream on the bus can be interpreted in various ways. For instance the sequence can be 2^n data bytes followed by 2^m ECC bytes (or vice versa), or the sequence can be 2^k iterations of 8 data bytes plus 1 ECC byte. Other information, such an information used by a directory-based cache coherence scheme, can also be managed this way. See, for example, Anant Agarwal, et al., "Scale Directory Schemes for Cache Consistency," 15th *International Symposium on Computer Architecture*, June 1988, pp. 280–289. Those skilled in the art will recognize alternative methods of implementing ECC schemes that are within the teachings of this invention.

Low Power 3-D Packaging

Another major advantage of this invention is that it drastically reduces the memory system power consumption. Nearly all the power consumed by a prior art DRAM is dissipated in performing row access. By using a single row access in a single RAM to supply all the bits for a block request (cared to a row-access in each of multiple RAMs in conventional memory systems) the power per bit can be made very small. Since the is power dissipated by memory devices using this invention is significantly reduced, the devices potentially can be placed much closer together than with conventional designs.

The bus architecture of this invention makes possible an innovative 3-D packaging technology. By using a narrow, multiplexed (time-shared) bus, the pin count for an arbitrarily large memory device can be kept quite small—on the order of 20 pins. Moreover, this pin count can be kept constant from one generation of DRAM density to the next. The low power dissipation allows each package to be smaller, with narrower pin pitches (spacing between the IC pins). With current surface mount technology supporting pin pitches as low as 20 mils, all off-device connections can be implemented on a single edge of the memory device. Semiconductor die useful in this invention preferably have connections or pads along one edge of the die which can then be wired or otherwise connected to the package pins with wires having similar lengths. This geometry also allows for very short leads, preferably with an effective lead length of less than 4 mm. Furthermore, this invention uses only bused interconnections, i.e., each pad on each device is connected by the bus to the corresponding pad of each other device.

The use of a low pin count and an edge-connected bus permits a simple 3-D package, whereby the devices are stacked and the bus is connected along a single edge of the stack. The fact that all of the signals are bused is important for the implementation of a simple 3-D structure. Without this, the complexity of the "backplane" would be too difficult to make cost effectively with current technology. The individual devices in a stack of the present invention can be packed quite tightly because of the low power dissipated by the entire memory system, permitting the devices to be stacked bumper-to-bumper or top to bottom. Conventional plastic-injection molded small outline (SO) packages can be used with a pitch of about 2.5 mm (100 mile), but the ultimate limit would be the device die thickness, which is about an order of magnitude smaller, 0.2–0.5 using current wafer technology.

Bus Electrical Description

By using devices with very low power dissipation and close physical packing, the bus can be made quite short, which in turn allows for short propagation times and high data rates. The bus of a preferred embodiment of the present invention consists of a set of resistor-terminated controlled impedance transmission lines which can operate up to a data rate of 500 MHz (2 ns cycles). The characteristics of the transmission lines are strongly affected by the loading caused by the DRAMs (or other slaves) mounted on the bus. These devices add lumped capacitance to the lines which both lowers the impedance of the lines and, decreases the transmission speed. In the loaded environment, the bus impedance is likely to be on the order of 25 ohms and the propagation velocity about c/4 (c=the speed of light) or 7.5 cm/ns. To operate at a 2 ns data rate, the transit time on the bus should preferably be kept under 1 ns, to leave 1 ns for the setup and hold time of the input receivers (described below) plus clock skew. Thus the bus lines must be kept quite short, under about 8 cm for maximum performance. Lower performance systems may have much longer lines, e.g. a 4 ns bus may have 24 cm lines (3 ns transit time, 1 ns setup and hold time).

In the preferred embodiment, the bus uses current source drivers. Each output must be able to sink 50 mA, which provides an output swing of about 500 mV or more. In the preferred embodiment of this invention, the bus is active low. The unasserted state (the high value) is preferably considered a logical zero, and the asserted value (low state) is therefore a logical 1. Those skilled in the art understand that the method of this invention can also be implemented using the opposite logical relation to voltage. The value of the unasserted state is set by the voltage on the termination resistors, and should be high enough to allow the outputs to act as current sources, while being as low as possible to reduce power dissipation. These constraints may yield a termination voltage about 2V above ground in the preferred implementation. Current source drivers cause the output voltage to be proportional to the sum of the sources driving the bus.

Referring to FIGS. 7a and 7b although there is no stable condition where two devices drive the bus at the same time, conditions can arise because of propagation delay on the wires where one device, A 41, can start driving its part of the bus 44 while the bus is still being driven by another device, B 42 (already asserting a logical 1 on the bus). in a system using current drivers, when B 42 is driving the bus (before time 46), the value at points 44 and 45 is logical 1. If B 42 switches off at time 46 just when A 41 switches on, the additional drive by device A 41 causes the voltage at the output 44 of A 41 to drop briefly below the normal value. The voltage returns to its normal value at time 47 when the effect of device B 42 turning off is felt. The voltage at point 45 goes to logical 0 when device B 42 turns off, then drops at time 47 when the effect of device A 41 turning on is felt. Since the logical 1 driven by current from device A 41 is propagated irrespective of the previous value on the bus, the value on the bus is guaranteed to settle after one time of flight (t_f) delay, that is, the time it takes a signal to propagate from one end of the bus to the other. If a voltage drive was used (as in ECL wired-ORing), a logical 1 on the bus (from device B 42 being previously driven) would prevent the transition put out by device A 41 being felt at the most remote part of the system, e.g., device 43, until the turnoff waveform from device B 42 reached device A 41 plus one time of flight delay, giving a worst case settling time of twice the time of flight delay.

Clocking

Clocking a high speed bus accurately without introducing error due to propagation delays can be implemented by having each device monitor two bus clock signals and then derive internally a device clock, the true system clock. The bus clock information can be sent on one or two lines to

provide a mechanism for each bused device to generate an internal device clock with zero skew relative to all the other device clocks. Referring to FIG. 8a, in the preferred implementation, a bus clock generator 50 at one end of the bus propagates an early bus clock signal in one direction along the bus, for example on line 53 from right to left, to the far end of the bus. The same clock signal then in passed through the direct connection shown to a second line 54, and returns as a late bus clock signal along the bus from the far end to the origin, propagating from left to right. A single bus clock line can be used if it is left unterminated at the far end of the bus, allowing the early bus clock signal to reflect back along the same line as a late bus clock signal.

FIG. 8b illustrates how each device 51, 52 receives each of the two bus clock signals at a different time (because of is propagation delay along the wires), with constant midpoint in time between the two bus clocks along the bus. At each device 51, 52, the rising edge 55 of Clock1 53 is followed by the rising edge 56 of Clock2 54. Similarly, the falling edge 57 of Clock1 53 is followed by the falling edge 58 of Clock2 54. This waveform relationship is observed at all other devices along the bus. Devices which are closer to the clock generator have a greater separation between Clock1 and Clock2 relative to devices farther from the generator because of the longer time required for each clock pulse to traverse the bus and return along line 54, but the midpoint in time 59, 60 between corresponding rising or falling edges in fixed because, for any given device, the length of each clock line between the far end of the bus and that device is equal. Each device must sample the two bus clocks and generate its own internal device clock at the midpoint of the two.

Clock distribution problem can be further reduced by using a bus clock and device clock rate equal to the bus cycle data rate divided by two, that is, the bus clock period is twice the bus cycle period. Thus a 500 MHz bus preferably uses a 250 MHz clock rate. This reduction in frequency provides two benefits. First it makes all signals on the bus have the same worst case data rates—data on a 500 MHz bus can only change every 2 ns. Second, clocking at half the bus cycle data rate makes the labeling of the odd and even bus cycles trivial, for example, by defining even cycles to be those when the internal device clock is 0 and odd cycles when the internal device clock is 1.

Multiple Buses

The limitation on bus length described above restricts the total number of devices that can be placed on a single bus. Using 2.5 spacing between devices, a single 8 cm bus will hold about 32 devices. Persons skilled in the art will recognize certain applications of the present invention wherein the overall data rate on the bus is adequate but memory or processing requirements necessitate a much larger number of devices (many more than 32). Larger systems can easily be built using the teachings of this invention by using one or more memory subsystems, designated primary bus units, each of which consists of two or more devices, typically 32 or close to the maximum allowed by bus design requirements, connected to a transceiver device.

Referring to FIG. 9, each primary bus unit can be mounted on a single circuit board 66, sometimes called a memory stick. Each transceiver device 19 in turn connects to a transceiver bus 65, similar or identical in electrical and other respects to the primary bus 18 described at length above. In a preferred implementation, all masters are situated on the transceiver bus so there are no transceiver delays between masters and all memory devices are on primary bus units so

that all memory accesses experience an equivalent transceiver delay, but persons skilled in the art will recognize how to implement systems which have masters on more than one bus unit and memory devices on the transceiver bus as well as on primary bus units. In general, each teaching of this invention which refers to a memory device can be practiced using a transceiver device and one or more memory devices on an attached primary bus unit. Other devices, generically referred to as peripheral devices, including disk controllers, video controllers or I/O devices can also be attached to either the transceiver bus or a primary bus unit, as desired. Persons skilled in the art will recognize how to use a single primary bus unit or multiple primary bus units needed with a transceiver bus in certain system designs.

The transceivers are quite simple in function. They detect request packets on the transceiver bus and transmit them to their primary bus unit. If the request packet calls for a write to a device on a transceiver's primary bus unit, that transceiver keeps track of the access time and block size and forwards all data from the transceiver bus to the primary bus unit during that time. The transceivers also watch their primary bus unit, forwarding any data that occurs there to the transceiver bus. The high speed of the buses means that the transceivers will need to be pipelined, and will require an additional one or two cycle delay for data to pass through the transceiver in either direction. Access times stored in masters on the transceiver bus must be increased to account for transceiver delay but access times stored in slaves on a primary bus unit should not be modified.

Persons skilled in the art will recognize that a more sophisticated transceiver can control transmissions to and from primary bus units. An additional control line, TmcvrRW can be bused to all devices on the transceiver bus, using that line in conjunction with the AddrValid line to indicate to all devices on the transceiver bus that the information on the data lines is: 1) a request packet, 2) valid data to a slave, 3) valid data from a slave, or 4) invalid data (or idle bus). Using this extra control line obviates the need for the transceivers to keep track of when data needs to be forwarded from its primary to the transceiver bus—all transceivers send all data from their primary bus to the transceiver bus whenever the control signal indicates condition 2) above. In a preferred implementation of this invention, if AddrValid and TmcvrRW are both low, there is no bus activity and the transceivers should remain in an idle state. A controller sending a request packet will drive AddrValid high, indicating to all devices on the transceiver bus that a request packet is being sent which each transceiver should forward to its primary bus unit. Each controller seeking to rite to a slave should drive both AddrValid and TmcrRW high, indicating valid data for a slave is present on the data lines. Each transceiver device will then transmit all data from the transceiver bus lines to each primary bus unit. Any controller expecting to receive information from a slave should also drive the TmcvrRW line high, but not drive AddrValid, thereby indicating to each transceiver to transmit any data coming from any slave on its primary local bus to the transceiver bus. A still more sophisticated transceiver would recognize signals addressed to or coming from its primary bus unit and transmit signals only at requested times.

An example of the physical mounting of the transceivers is shown in FIG. 9. One important feature of this physical arrangement is to integrate the bus of each transceiver 19 with the original bus of DRAMs or other devices 15, 16, 17 on the primary bus unit 66. The transceivers 19 have pins on two sides, and are preferably mounted flat on the primary

bus unit with a first set of pins connected to primary bus 18. A second set of transceiver pins 20, preferably orthogonal to the first set of pins, are oriented to allow the transceiver 19 to be attached to the transceiver bus 65 in much the same way as the DRAMs were attached to the primary bus unit. The transceiver bus can be generally planar and in a different plane, preferably orthogonal to the plane of each primary bus unit. The transceiver bus can also be generally circular with primary bus units mounted perpendicular and tangential to the transceiver bus.

Using this two level scheme allows one to easily build a system that contains over 500 slaves (16 buses of 32 DRAMs each). Persons skilled in the art can modify the device ID scheme described above to accommodate more than 256 devices, for example by using a longer device ID or by using additional registers to hold some of the device ID. This scheme can be extended in yet a third dimension to make a second-order transceiver bus, connecting multiple transceiver buses by aligning transceiver bus units parallel to and on top of each other and busing corresponding signal lines through a suitable transceiver. Using such a second-order transceiver bus, one could connect many thousands of slave devices into what is effectively a single bus.

Device Interface

The device interface to the high-speed bus can be divided into three main parts. The first part is the electrical interface. This part includes the input receivers, bus drivers and clock generation circuitry. The second part contains the address comparison circuitry and timing registers. This part takes the input request packet and determines if the request is for this device, and if it is, starts the internal access and delivers the data to the pins at the correct time. The final part, specifically for memory devices such as DRAMs, is the DRAM column access path. This part needs to provide bandwidth into and out of the DRAM sense amps greater than the bandwidth provided by conventional DRAMs. The implementation of the electrical interface and DRAM column access path are described in more detail in the following sections. Persons skilled in the art recognize how to modify prior-art address comparison circuitry and prior-art register circuitry in order to practice the present invention.

Electrical Interface—Input/Output Circuitry

A block diagram of the preferred input/output circuit for address/data/control lines is shown in FIG. 10. This circuitry is particularly well-suited for use in DRAM devices but it can be used or modified by one skilled in the art for use in other devices connected to the bus of this invention. It consists of a set of input receivers 71, 72 and output driver 76 connected to input/output line 69 and pad 75 and circuitry to use the internal clock 73 and internal clock complement 74 to drive the input interface. The clocked input receivers take advantage of the synchronous nature of the bus. To further reduce the performance requirements for device input receivers, each device pin, and thus each bus line, is connected to two clocked receivers, one to sample the even cycle inputs, the other to sample the odd cycle inputs. By thus de-multiplexing the input 69 at the pin, each clocked amplifier is given a full 2 ns cycle to amplify the bus low-voltage-swing signal into a full value CMOS logic signal. Persons skilled in the art will recognize that additional clocked input receivers can be used within the teachings of this invention. For example, four input receivers could be connected to each device pin and clocked by a modified internal device clock to transfer sequential bits from the bus to internal device circuits, allowing still higher external bus speeds or still longer settling times to amplify the bus low-voltage-swing signal into a full value CMOS logic signal.

The output drivers are quite simple, and consist of a single RHOS pulldown transistor 76. This transistor is sized so that under worst case conditions it can still sink the 50 mA required by the bus. For 0.8 micron CMOS technology, the transistor will need to be about 200 microns long. Overall bus performance can be improved by using feedback techniques to control output transistor current so that the current through the device is roughly 50 mA under all operating conditions, although this is not absolutely necessary for proper bus operation. An example of one of many methods known to persons skilled in the art for using feedback techniques to control current is described in Hans Schumacher, et al., CMOS Subnanosecond True-ECL Output Buffer," *J. Solid State Circuits*, Vol. 25 (1), pp. 150–154 (February 1990). Controlling this current improves performance and reduces power dissipation. This output driver which can be operated at 500 Hz, can in turn be controlled by a suitable multiplexer with two or more (preferably four) inputs connected to other internal chip circuitry, all of which can be designed according to well known prior art.

The input receivers of every slave must be able to operate during every cycle to determine whether the signal on the bus is a valid request packet. This requirement leads to a number of constraints on the input circuitry. In addition to requiring small acquisition and resolution delays, the circuits must take little or no DC power, little AC power and inject very little current back into the input or reference lines. The standard clocked DRAM sense amp shown in FIG. 11 satisfies all these requirements except the need for low input currents. When this sense amp goes from sense to sample, the capacitance of the internal nodes 93 and 84 in FIG. 11 is discharged through the reference line 68 and input 69, respectively. This particular current is small, but the sum of such currents from all the inputs into the reference lines summed over all devices can be reasonably large.

The fact that the sign of the current depends upon on the previous received data makes matters worse. One way to solve this problem is to divide the sample period into two phases. During the first phase, the inputs are shorted to a buffered version of the reference level (which may have an offset). During the second phase, the inputs are connected to the true inputs. This scheme does not remove the input current completely, since the input must still charge nodes 83 and 84 from the reference value to the current input value, but it does reduce the total charge required by about a factor of 10 (requiring only a 0.25V change rather than a 2.5V change). Persons skilled in the art will recognize that many other methods can be used to provide a clocked amplifier that will operate on very low input currents.

One important part of the input/output circuitry generates an internal device clock based on early and late bus clocks. Controlling clock skew (the difference in clock timing between devices) is important in a system running with 2 ns cycles, thus the internal device clock is generated so the input sampler and the output driver operate as close in time as possible to midway between the two bus clocks.

A block diagram of the internal device clock generating circuit is shown in FIG. 12 and the corresponding timing diagram in FIG. 13. The basic idea behind this circuit is relatively simple. A DC amplifier 102 is used to convert the small-swing bus clock into a full-swing CMOS signal. This signal is then fed into a variable delay line 103. The output of delay line 103 feeds three additional delay lines: 104 having a fixed delay; 105 having the same fixed delay plus a second variable delay; and 106 having the same fixed delay plus one half of the second variable delay. The outputs 107, 108 of the delay lines 104 and 105 drive clocked input

receivers 101 and 111 connected to early and late bus clock inputs 100 and 110, respectively. These input receivers 101 and 111 have the same design as the receivers described above and shown in FIG. 11. Variable delay lines 103 and 105 are adjusted via feedback lines 116, 115 so that input receivers 101 and 111 sample the bus clocks just as they transition. Delay lines 103 and 105 are adjusted so that the falling edge 120 of output 107 precedes the falling edge 121 of the early bus clock, Clock1 53, by an amount of time 128 equal to the delay in input sampler 101. Delay line 108 is adjusted in the same way so that falling edge 122 precedes the falling edge 123 of late bus clock, Clock2 54, by the delay 128 in input sampler 111.

Since the outputs 107 and 108 are synchronized with the two bus clocks and the output 73 of the last delay line 106 is midway between outputs 107 and 108, that is, output 73 follows output 107 by the same amount of time 129 that output 73 precedes output 108, output 73 provides an internal device clock midway between the bus clocks. The falling edge 124 of internal device clock 73 precedes the time of actual input sampling 125 by one sampler delay. Note that this circuit organization automatically balances the delay in substantially all device input receivers 71 and 72 (FIG. 10), since outputs 107 and 108 are adjusted so the bus clocks are sampled by input receivers 101 and 111 just as the bus clocks transition.

In the preferred embodiment, two sets of these delay lines are used, one to generate the true value of the internal device clock 73, and the other to generate the complement 74 without adding any inverter delay. The dual circuit allows generation of truly complementary clocks, with extremely small skew. The complement internal device clock is used to clock the 'even' input receivers to sample at time 127, while the true internal device clock is used to clock the 'odd' input receivers to sample at time 125. The true and complement internal device clocks are also used to select which data is driven to the output drivers. The gate delay between the internal device clock and output circuits driving the bus in slightly greater than the corresponding delay for the input circuits, which means that the new data always will be driven on the bus slightly after the old data has been sampled.

DRAM Column Access Modification

A block diagram of a conventional 4 MBit DRAM 130 is shown in FIG. 15. The DRAM memory array is divided into a number of subarrays 150–157, for example, 8. Each subarray is divided into arrays 148, 149 of memory cells. Row address selection in performed by decoders 146. A column decoder 147A, 147B, including column sense amps on either side of the decoder, runs through the core of each subarray. These column sense amps can be set to precharge or latch the most-recently stored value, as described in detail above. Internal I/O lines connect each set of sense-amps, as gated by corresponding column decoders, to input and output circuitry connected ultimately to the device pins. These internal I/O lines are used to drive the data from the selected bit lines to the data pins (some of pins 131–145), or to take the data from the pins and write the selected bit lines. Such a column access path organized by prior art constraints does not have sufficient bandwidth to interface with a high speed bus. The method of this invention does not require changing the overall method used for column access, but does change implementation details. Many of these details have been implemented selectively in certain fast memory devices abut never in conjunction with the bus architecture of this invention.

Running the internal I/O lines in the conventional way at high bus cycle rates is not possible. In the preferred method,

several (preferably 4) bytes are read or written during each cycle and the column access path is modified to run at a lower rate (the inverse of the number of bytes accessed per cycle, preferably ¼ of the bus cycle rate). Three different techniques are used to provide the additional internal I/O lines required and to supply data to memory cells at this rate. First, the number of I/O bit lines in each subarray running through the column decoder 147A, B is increased, for example, to 16, eight for each of the two columns of column sense amps and the column decoder selects one set of columns from the "top" half 148 of subarray 150 and one set of columns from the "bottom" half 149 during each cycle, where the column decoder selects one column sense amp per I/O bit line. Second, each column I/O line is divided into two halves, carrying data independently over separate internal I/O lines from the left half 147A and right half 147B of each subarray (dividing each subarray into quadrants) and the column decoder selects sense amps from each right and left half of the subarray, doubling the number of bits available at each cycle. Thus each column decode selection turns on n column sense amps, where n equals four (top left and right, bottom left and right quadrants) times the number of I/O lines in the bus to each subarray quadrant (8 lines each×4=32 lines in the preferred implementation). Finally, during each RAS cycle, two different subarrays, e.g. 157 and 153, are accessed. This doubles again the available number of I/O lines containing data. Taken together, these changes increase the internal I/O bandwidth by at least a factor of 8. Four internal buses are used to route these internal I/O lines. Increasing the number of I/O lines and then splitting them in the middle greatly reduces the capacitance of each internal I/O line which in turn reduces the column access time, increasing the column access bandwidth even further.

The multiple, gated input receivers described above allow high speed input from the device pins onto the internal I/O lines and ultimately into memory. The multiplexed output driver described above is used to keep up with the data flow available using these techniques. Control means are provided to select whether information at the device pins should be treated as an address, and therefore to be decoded, or input or output data to be driven onto or read from the internal I/O lines.

Each subarray can access 32 bits per cycle, 16 bits from the left subarray and 16 from the right subarray. With 8 I/O lines per sense-amplifier column and accessing two subarrays at a time, the DRAM can provide 64 bits per cycle. This extra I/O bandwidth is not needed for reads (and is probably not used), but may be needed for writes. Availability of write bandwidth is a more difficult problem than read bandwidth because over-writing a value in a sense-amplifier may be a slow operation, depending on how the sense amplifier is connected to the bit line. The extra set of internal I/O lines provides some bandwidth margin for write operations.

Persons skilled in the art will recognize that many variations of the teachings of this invention can be practiced that still fall within the claims of this invention which follow.

What is claimed is:

1. A method of controlling a memory device by a memory controller, wherein the memory device includes a plurality of memory cells, the method of controlling the memory device comprises:

    providing first block size information to the memory device, wherein the memory device is capable of processing the first block size information, wherein the first block size information is provided by the memory controller and is representative of a first amount of data to be input by the memory device; and

issuing a first operation code to the memory device, wherein in response to the first operation code, the memory device inputs the first amount of data.

2. The method of claim 1 wherein the memory device inputs the first amount of data synchronously with respect to an external clock signal.

3. The method of claim 1 further including:

providing second block size information to the memory device, wherein the second block size information defines a second amount of data to be input by the memory device; and

issuing a second operation code to the memory device, wherein in response to the second operation code, the memory device inputs the second amount of data.

4. The method of claim 1 wherein the first block size information and the first operation code are included in a request packet.

5. The method of claim 4 wherein the first block size information and the first operation code are included in the same request packet.

6. The method of claim 1 further including providing the first amount of data to the memory device.

7. The method of claim 6 wherein the first amount of data is provided to the memory device after a delay time transpires.

8. The method of claim 7 wherein the delay time is representative of a number of clock cycles of an external clock signal.

9. The method of claim 1 in wherein the first block size information is a binary representation of the first amount of data.

10. The method of claim 1 wherein the first amount of data is output, by the memory controller, synchronously with respect to an external clock signal and during a plurality of clock cycles of the external clock signal.

11. The method of claim 1 wherein the first operation code is issued onto a bus.

12. The method of claim 11 wherein the bus includes a plurality of signal lines to multiplex control information, address information and data.

13. The method of claim 1 further including providing address information to the memory device.

14. A method of operation in a synchronous memory device, wherein the memory device includes a plurality of memory cells, the method of operation of the memory device comprises:

receiving first block size information from a memory controller, wherein the memory device is capable of processing the first block size information, wherein the first block size information represents a first amount of data to be input by the memory device in response to an operation code;

receiving the operation code, from the memory controller, synchronously with respect to an external clock signal; and

inputting the first amount of data in response to the operation code.

15. The method of claim 14 wherein inputting the first amount of data includes receiving the first amount of data synchronously with respect to the external clock signal.

16. The method of claim 15 wherein the first amount of data is sampled over a plurality of clock cycles of the external clock signal.

17. The method of claim 14 wherein the first block size information and the operation code are included in a request packet.

18. The method of claim 17 wherein the first block size information and the operation code are included in the same request packet.

19. The method of claim 14 wherein the first block size information is a binary representation of the first amount of data to be input in response to the operation code.

20. The method of claim 11 wherein the first amount of data is output, by the memory controller, synchronously during a plurality of clock cycles of the external clock signal.

21. The method of claim 14 further including generating an internal clock signal, using a delay locked loop and the external clock signal wherein the first amount of data is input synchronously with respect to the internal clock signal.

22. The method of claim 14 further including generating first and second internal clock signals using clock generation circuitry and the external clock signal, wherein the first amount of data is input synchronously with respect to the first and second internal clock signals.

23. The method of claim 22 wherein the first and second internal clock signals are generated by a delay lock loop.

24. The method of claim 14 wherein the operation code, the first block size information and address information are included in a packet.

25. The method of claim 14 further including receiving address information from the memory controller.

26. The method of claim 14 wherein the first block size information, and the operation code are received from an external bus.

27. The method of claim 26 wherein the first block size information, and the operation code are received from the same external bus.

28. The method of claim 27 wherein the external bus is used to multiplex address information, control information and data.

29. A method of operation of an integrated circuit, wherein the integrated circuit includes a dynamic random access memory array having a plurality of memory cells, the method of operation comprises:

receiving block size information from a controller, memory device is capable of processing the first block size information wherein the block size information represents an amount of data to be input in response to an operation code;

receiving the operation code from the controller; and

inputting the amount of data in response to the operation code.

30. The method of claim 29 further including storing the amount of data in the memory array.

31. The method of claim 29 wherein the block size information and the operation code are included in a request packet.

32. The method of claim 29 wherein the block size information is a binary representation of the amount of data to be input in response to the operation code.

33. The method of claim 29 wherein the amount of data is input, in response to the operation code, after a delay time transpires.

34. The method of claim 33 wherein the delay, time is representative of a number of clock cycles of the external clock signal.

35. The method of claim 29 further including receiving address information from the controller.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,452,863 B2

DATED : September 17, 2002

INVENTOR(S) : Farmwald et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 26,

Line 7, delete "11" and substitute -- 14 --.

Line 43, insert -- wherein the -- before "memory device".

Line 61, delete "," appearing between "delay" and "time".

Signed and Sealed this

First Day of April, 2003

JAMES E. ROGAN
Director of the United States Patent and Trademark Office